Development System Reference Guide

8.2i

XILINX®

About This Guide

The *Development System Reference Guide* contains information about the command line software programs in the Xilinx Development System. Most chapters are organized as follows:

- A brief summary of program functions
- A syntax statement
- A description of the input files used and the output files generated by the program
- A listing of the commands, options, or parameters used by the program
- Examples of how to use the program

For an overview of the Xilinx Development System describing how these programs are used in the design flow, see Chapter 2, "Design Flow".

Guide Contents

The *Development System Reference Guide* provides detailed information about converting, implementing, and verifying designs with the Xilinx command line tools. Check the program chapters for information on what program works with each family of Field Programmable Gate Array (FPGA) or Complex Programmable Logic Device (CPLD). Following is a brief overview of the contents and organization of the *Development System Reference Guide*:

Note: For information on timing constraints, UCF files, and PCF files, see the *Constraints Guide*.

- Chapter 1, "Introduction" —This chapter describes some basics that are common to the different Xilinx Development System modules.
- Chapter 2, "Design Flow"—This chapter describes the basic design processes: design entry, synthesis, implementation, and verification.
- Chapter 3, "Tcl"—Tcl is designed to complement and extend the graphical user interface (GUI). Xilinx Tcl commands provide a batch interface that makes it convenient to execute the exact same script or steps over and over again.
- Chapter 4, "PARTGen"—PARTGen allows you to obtain information about installed devices and families.
- Chapter 5, "Logical Design Rule Check"—The Logical Design Rule Check (DRC) comprises a series of tests run to verify the logical design described by the Native Generic Database (NGD) file.
- Chapter 6, "NGDBuild"—NGDBuild performs all of the steps necessary to read a netlist file in EDIF format and create an NGD (Native Generic Database) file describing the logical design reduced to Xilinx primitives.

- Chapter 7, "MAP"—MAP packs the logic defined by an NGD file into FPGA elements such as CLBs, IOBs, and TBUFs.
- Chapter 8, "Physical Design Rule Check"—The physical Design Rule Check (DRC) comprises a series of tests run to discover physical errors in your design.
- Chapter 9, "PAR"—PAR places and routes FPGA designs.
- Chapter 10, "XPower"—XPower is a power and thermal analysis tool that generates power and thermal estimates after the PAR or CPLDfit stage of the design.
- Chapter 11, "PIN2UCF,"—PIN2UCF generates pin-locking constraints in a UCF file by reading a a placed NCD file for FPGAs or GYD file for CPLDs.
- Chapter 12, "TRACE"—Timing Reporter and Circuit Evaluator (TRACE) performs static timing analysis of a physical design based on input timing constraints.
- Chapter 13, "Speedprint"— Speedprint lists block delays for a specified device and its speed grades.
- Chapter 14, "BitGen"—BitGen creates a configuration bitstream for an FPGA design.
- Chapter 15, "BSDLAnno"—BSDLAnno automatically modifies a BSDL file for post-configuration interconnect testing.
- Chapter 16, "PROMGen" —PROMGen converts a configuration bitstream (BIT) file into a file that can be downloaded to a PROM. PROMGen also combines multiple BIT files for use in a daisy chain of FPGA devices.
- Chapter 17, "IBISWriter"—IBISWriter creates a list of pins used by the design, the signals inside the device that connect those pins, and the IBIS buffer model that applies to the IOB connected to the pins.
- Chapter 18, "CPLDfit" —CPLDfit reads in an NGD file and fits the design into the selected CPLD architecture.
- Chapter 19, "TSIM" — TSIM formats an implemented CPLD design (VM6) into a format usable by the NetGen timing simulation flow, which produces a back-annotated timing file for simulation.
- Chapter 20, "TAEngine" —TAEngine performs static timing analysis on a successfully implemented Xilinx CPLD design (VM6).
- Chapter 21, "Hprep6" —Hprep6 takes an implemented CPLD design (VM6) from CPLDfit and generates a JEDEC (JED) programming file.
- Chapter 22, "NetGen"—NetGen reads in applicable Xilinx implementation files, extracts design data, and generates netlists that are used with supported third-party simulation, equivalence checking, and static timing analysis tools.
- Chapter 23, "XFLOW"—XFLOW automates the running of Xilinx implementation and simulation flows.
- Chapter 24, "Data2MEM"—Data2MEM transforms CPU execution code, or pure data, into Block RAM initialization records.
- "Appendix A"—This appendix gives an alphabetic listing of the files used by the Xilinx Development System.
- "Appendix B" —This appendix describes the netlist reader, EDIF2NGD, and how it interacts with NGDBuild.

Additional Resources

To find additional documentation, see the Xilinx website at:

http://www.xilinx.com/literature.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

http://www.xilinx.com/support.

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *design_name*
Helvetica bold	Commands that you select from a menu	**File → Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *design_name*
	References to other manuals	See the *Development System Reference Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	**`ngdbuild`** [*`option_name`*] *`design_name`*
Braces { }	A list of items from which you must choose one or more	**`lowpwr =`**{**`on`**|**`off`**}
Vertical bar |	Separates items in a list of choices	**`lowpwr =`**{**`on`**|**`off`**}

Convention	Meaning or Use	Example

Vertical ellipsis
.
.
.
	Repetitive material that has been omitted	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Horizontal ellipsis `...`	Repetitive material that has been omitted	**`allow block`** *block_name loc1 loc2 ... locn;*

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current file or in another file in the current document	See the section "Additional Resources" for details. Refer to "Title Formats" in Chapter 1 for details.
Red text	Cross-reference link to a location in another document	See Figure 2-5 in the *Virtex-II Handbook*.
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.

Table of Contents

Chapter 3: Tcl

Chapter 4: PARTGen

Chapter 5: Logical Design Rule Check

Chapter 6: NGDBuild

Chapter 7: MAP

Chapter 10: XPower

Chapter 11: PIN2UCF

Chapter 12: TRACE

Chapter 13: Speedprint

Chapter 14: BitGen

Chapter 15: BSDLAnno

Chapter 16: PROMGen

Chapter 17: IBISWriter

Chapter 18: CPLDfit

Chapter 19: TSIM

Chapter 20: TAEngine

Chapter 21: Hprep6

Chapter 22: NetGen

Chapter 23: XFLOW

Chapter 24: Data2MEM

XILINX®

Chapter 1

Introduction

This chapter describes the command line programs for the Xilinx development system. The chapter contains the following sections:

- "Command Line Program Overview"
- "Command Line Syntax"
- "Command Line Options"
- "Invoking Command Line Programs"

Command Line Program Overview

Xilinx command line programs allow you to implement and verify your design. The following table lists the programs you can use for each step in the design flow. For detailed information, see Chapter 2, "Design Flow".

Table 1-1: **Command Line Programs in the Design Flow**

Design Flow Step	Command Line Program
Design Implementation	NGDBuild, MAP, PAR, Xplorer, BitGen
Design Preservation	TCL
Timing Simulation and Back Annotation (Design Verification)	NetGen
Static Timing Analysis (Design Verification)	TRACE

You can run these programs in the standard design flow or use special options to run the programs for design preservation. Each command line program has multiple options, which allow you to control how a program executes. For example, you can set options to change output file names, to set a part number for your design, or to specify files to read in when executing the program. You can also use options to create guide files and run guide mode to maintain the performance of a previously implemented design.

Some of the command line programs described in this manual underlie many of the Xilinx Graphical User Interfaces (GUIs). The GUIs can be used in conjunction with the command line programs or alone. For information on the GUIs, see the online Help provided with each Xilinx tool.

Command Line Syntax

Command line syntax always begins with the command line program name. The program name is followed by any options and then file names. Use the following rules when specifying command line options:

- Enter options in any order.
- Precede options with a hyphen (-) and separate them with spaces.
- Be consistent with upper case and lower case.
- When an option requires a parameter, separate the parameter from the option by spaces or tabs. For example, the following shows the command line syntax for running PAR with the effort level set to medium:
  - Correct: **par -ol med**
  - Incorrect: **par -ol med**
- When using options that can be specified multiple times, precede the parameter with the option letter. In this example, the -l option shows the list of libraries to search:
  - Correct: **-l xilinxun -l synopsys**
  - Incorrect: **-l xilinxun synopsys**
- Enter parameters that are bound to an option *after* the option.
  - Correct: **-f** *command_file*
  - Incorrect: *command_file* **-f**

Use the following rules when specifying file names:

- Enter file names in the order specified in the chapter that describes the command line program. In this example the correct order is program, input file, output file, and then physical constraints file.
  - Correct: **par input.ncd output.ncd freq.pcf**
  - Incorrect: **par input.ncd freq.pcf output.ncd**
- Use lower case for all file extensions (for example, .ncd).

Command Line Options

The following options are common to many of the command line programs in the Xilinx Development System.

–f (Execute Commands File)

For any Xilinx Development System program, you can store command line program options and file names in a command file. You can then execute the arguments by entering the program name with the –f option followed by the name of the command file. This is useful if you frequently execute the same arguments each time you execute a program or if the command line command becomes too long.

You can use the file in the following ways:

- To supply all the command options and file names for the program, as in the following example:

  par -f *command_file*

  command_file is the name of the file that contains the command options and file names.

- To insert certain command options and file names within the command line, as in the following example:

  ```
  par -f placeoptions -s 4 -f routeoptions design_i.ncd design_o.ncd
  ```

  placeoptions is the name of a file containing placement command parameters.

  routeoptions is the name of a file containing routing command parameters.

You create the command file in ASCII format. Use the following rules when creating the command file:

- Separate program options and file names with spaces.
- Precede comments with the pound sign (#).
- Put new lines or tabs anywhere white space is allowed on the UNIX or DOS command line.
- Put all arguments on the same line, one argument per line, or a combination of these.
- All carriage returns and other non-printable characters are treated as spaces and ignored.
- No line length limitation exists within the file.

Following is an example of a command file:

```
#command line options for par for design mine.ncd
-n 10
-w
01 5
-s 2 #will save the two best results
/home/yourname/designs/xilinx/mine.ncd
#directory for output designs
/home/yourname/designs/xilinx/output.dir
#use timing constraints file
/home/yourname/designs/xilinx/mine.pcf
```

–h (Help)

When you enter a program name followed by **–help** or **–h**, a message displays that lists all the available options and their parameters as well as the file types for use with the program. The message also explains each of the options.

Following are descriptions for the symbols used in the help message:

Symbol	Description
[ ]	Encloses items that are optional.
{ }	Encloses items that may be repeated.
< >	Encloses a variable name or number for which you must substitute information.
,	Shows a range for an integer variable.
–	Shows the start of an option name.
:	Binds a variable name to a range.

Symbol	Description
|	Logical OR to show a choice of one out of many items. The OR operator may only separate logical groups or literal keywords.
( )	Encloses a logical grouping for a choice between subformats.

Following are examples of syntax used for file names:

- <*infile*[**.ncd**]> shows that typing the .ncd extension is optional but that the extension must be .ncd.
- <*infile*<**.edn**>> shows that the .edn extension is optional and is appended only if there is no other extension in the file name.

For architecture-specific programs, such as BitGen, you can enter the following to get a verbose help message for the specified architecture:

program_name **–h** *architecture_name*

You can redirect the help message to a file to read later or to print out by entering the following:

program_name **–h >** *filename*

On the UNIX command line, enter the following to redirect the help message to a file and return to the command prompt.

program_name **–h > &** *filename*

–intstyle (Integration Style)

You can limit screen output, based on the integration style that you are running, to warning and error messages only. When using the –intstyle option, one of three modes must be specified: *ise*, *xflow*, or *silent*. The mode sets the way information is displayed in the following ways:

-intstyle {**ise** | **xflow** | **silent**}

-intstyle ise

This mode indicates the program is being run as part of an integrated design environment.

-intstyle xflow

This mode indicates the program is being run as part of an integrated batch flow.

-intstyle silent

This mode limits screen output to warning and error messages only.

Note: The -intstyle option is automatically invoked when running in an integrated environment, such as Project Navigator or XFLOW.

–p (Part Number)

You can use the –p option with the EDIF2NGD, NGDBuild, MAP, and XFLOW programs to specify the part into which your design will be implemented. You can specify a part number at the following different points in the design flow:

- In the input netlist (does not require the –p option)
- In a Netlist Constraints File (NCF) (does not require the –p option)
- With the –p option when you run a netlist reader (EDIF2NGD) User Constraints File (UCF) (does not require the –p option)
- With the –p option when you run NGDBuild

  By the time you run NGDBuild, you must have already specified a device architecture.
- With the –p option when you run MAP

  When you run MAP, an architecture, device, and package must be specified, either on the MAP command line or earlier in the design flow. If you do not specify a speed, MAP selects a default speed. You can only run MAP using a part number from the architecture you specified when you ran NGDBuild.

Note: Part numbers specified in a later step of the design flow override a part number specified in an earlier step. For example, a part specified when you run MAP overrides a part specified in the input netlist.

A complete Xilinx part number consists of the following elements:

- Architecture (for example, Spartan-3e)
- Device (for example, xc3s100e)
- Package (for example, vq100)
- Speed (for example, -4)

Note: The Speedprint program lists block delays for device speed grades. The -s option allows you to specify a speed grade. If you do not specify a speed grade, Speedprint reports the default speed grade for the device you are targeting. See "–s (Speed Grade)" in Chapter 13 for details.

The following table lists multiple ways to specify a part on the command line.

Table 1-2: **Part Number Examples**

Specification	Examples
Architecture only	virtex virtex2 virtex2p virtex4 spartan2 spartan2e spartan 3 spartan 3e xc9500 xpla3
Device only	xc4vfx12 xc3s100e

Table 1-2: **Part Number Examples**

Specification	Examples
DevicePackage	xc4fx12sf363 xc3s100evq100
Device–Package	xc4vfx12-sf363 xc3s100e-vq100
DeviceSpeed–Package	xc4vfx1210-sf363 xc3s100e4-vq100
DevicePackage–Speed	xc4fx12sf363-10 xc3s100evq100-4
Device–Speed–Package	xc4vfx12-10-sf363 xc3s100e-4-vq100
Device–SpeedPackage	xc4vfx12-10sf363 xc3s100e-4vq100

Invoking Command Line Programs

You start Xilinx Development System command line programs by entering a command at the UNIX™ or DOS™ command line. See the program-specific chapters in this book for the appropriate syntax

Xilinx also offers the XFLOW program, which allows you to automate the running of several programs at one time. See Chapter 23, “XFLOW” for more information.

Chapter 2

Design Flow

This chapter describes the process for creating, implementing, verifying, and downloading designs for FPGA and CPLD devices. For a complete description of FPGAs and CPLDs, refer to the Xilinx Data Sheets at
http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp

This chapter contains the following sections:

Design Flow Overview

The standard design flow comprises the following steps:

1. Design Entry and Synthesis—In this step of the design flow, you create your design using a Xilinx-supported schematic editor, a hardware description language (HDL) for text-based entry, or both. If you use an HDL for text-based entry, you must synthesize the HDL file into an EDIF file or, if you are using the Xilinx Synthesis Technology (XST) GUI, you must synthesize the HDL file into an NGC file.
2. Design Implementation—By implementing to a specific Xilinx architecture, you convert the logical design file format, such as EDIF, that you created in the design entry and synthesis stage into a physical file format. The physical information is contained in the native circuit description (NCD) file for FPGAs and the VM6 file for CPLDs. Then you create a bitstream file from these files and optionally program a PROM or EPROM for subsequent programming of your Xilinx device.
3. Design Verification—Using a gate-level simulator or cable, you ensure that your design meets your timing requirements and functions properly. See the iMPACT online help for information about Xilinx download cables and demonstration boards.

The full design flow is an iterative process of entering, implementing, and verifying your design until it is correct and complete. The Xilinx Development System allows quick design iterations through the design flow cycle. Because Xilinx devices permit unlimited reprogramming, you do not need to discard devices when debugging your design in circuit.

The following figure shows the standard Xilinx design flow.

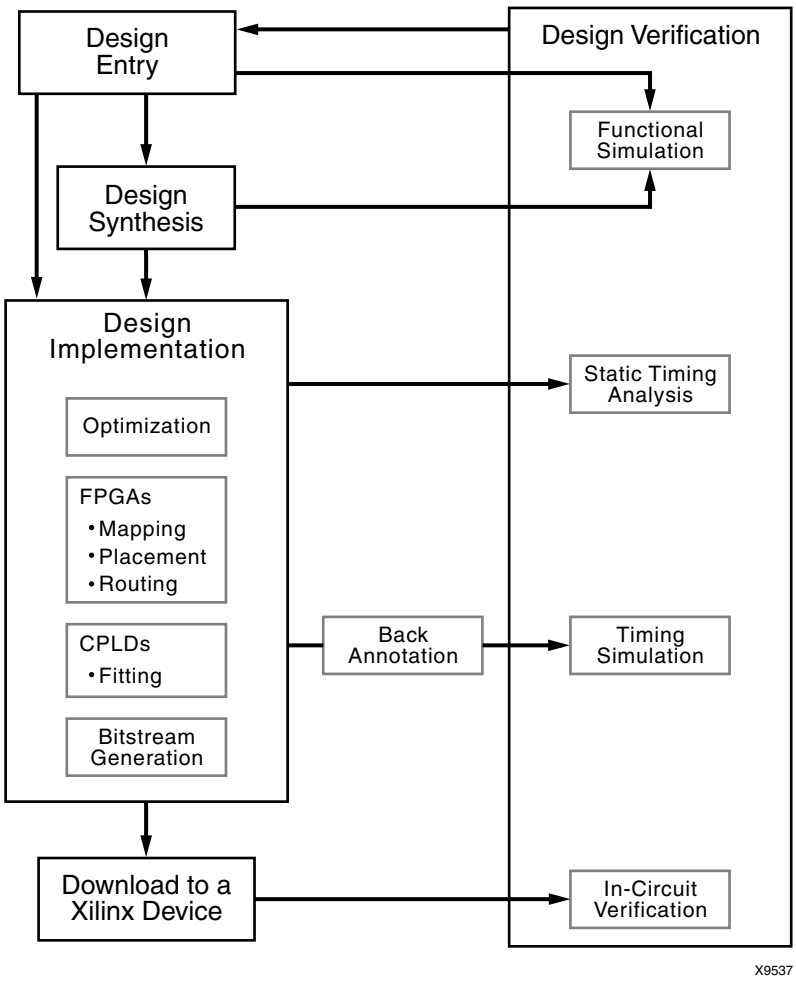


Figure 2-1: **Xilinx Design Flow**

The following figure shows the Xilinx *software* flow chart for FPGA designs.

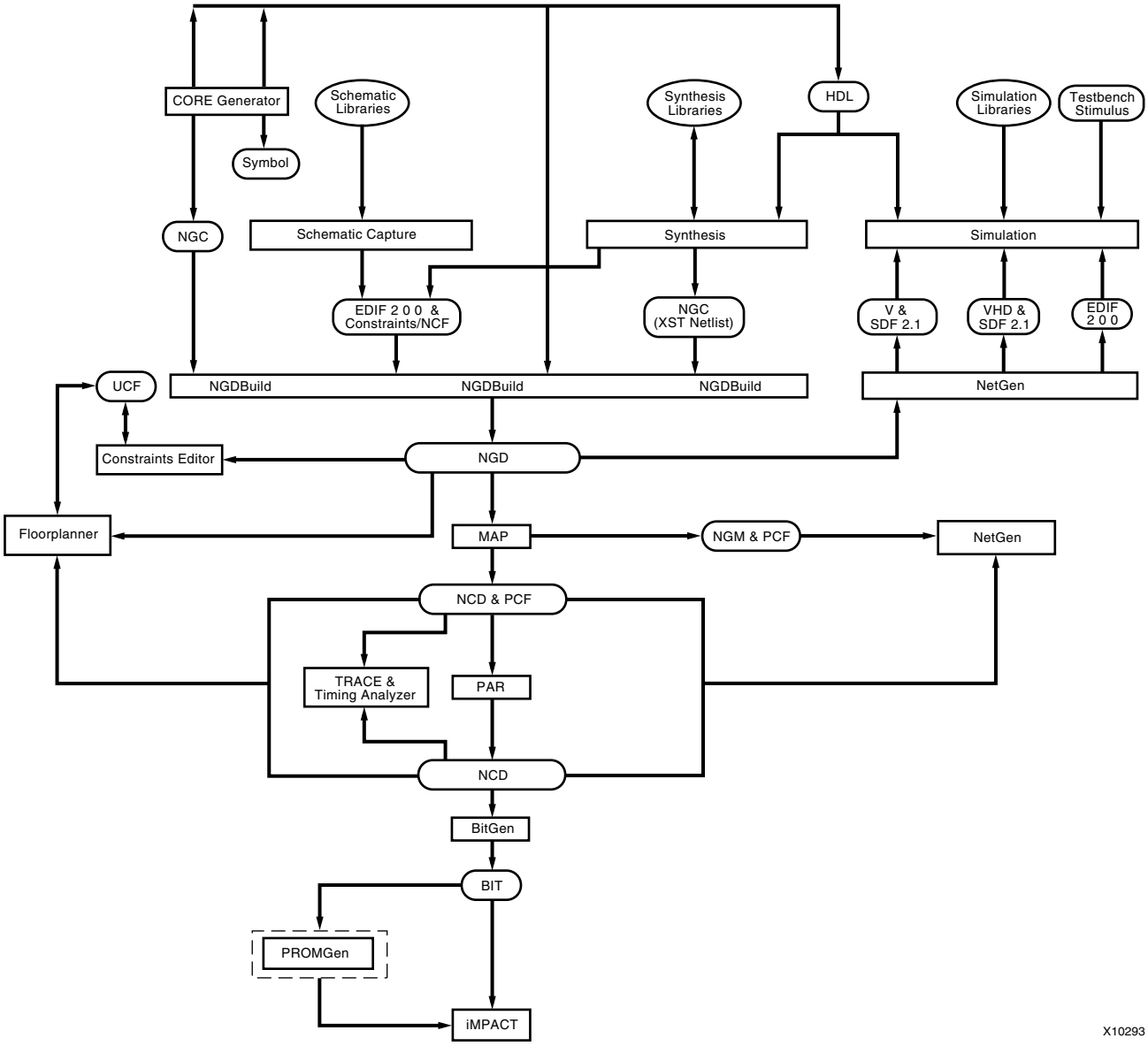


Figure 2-2: **Xilinx Software Design Flow (FPGAs)**

The following figure shows the Xilinx *software* flow chart for CPLD designs.

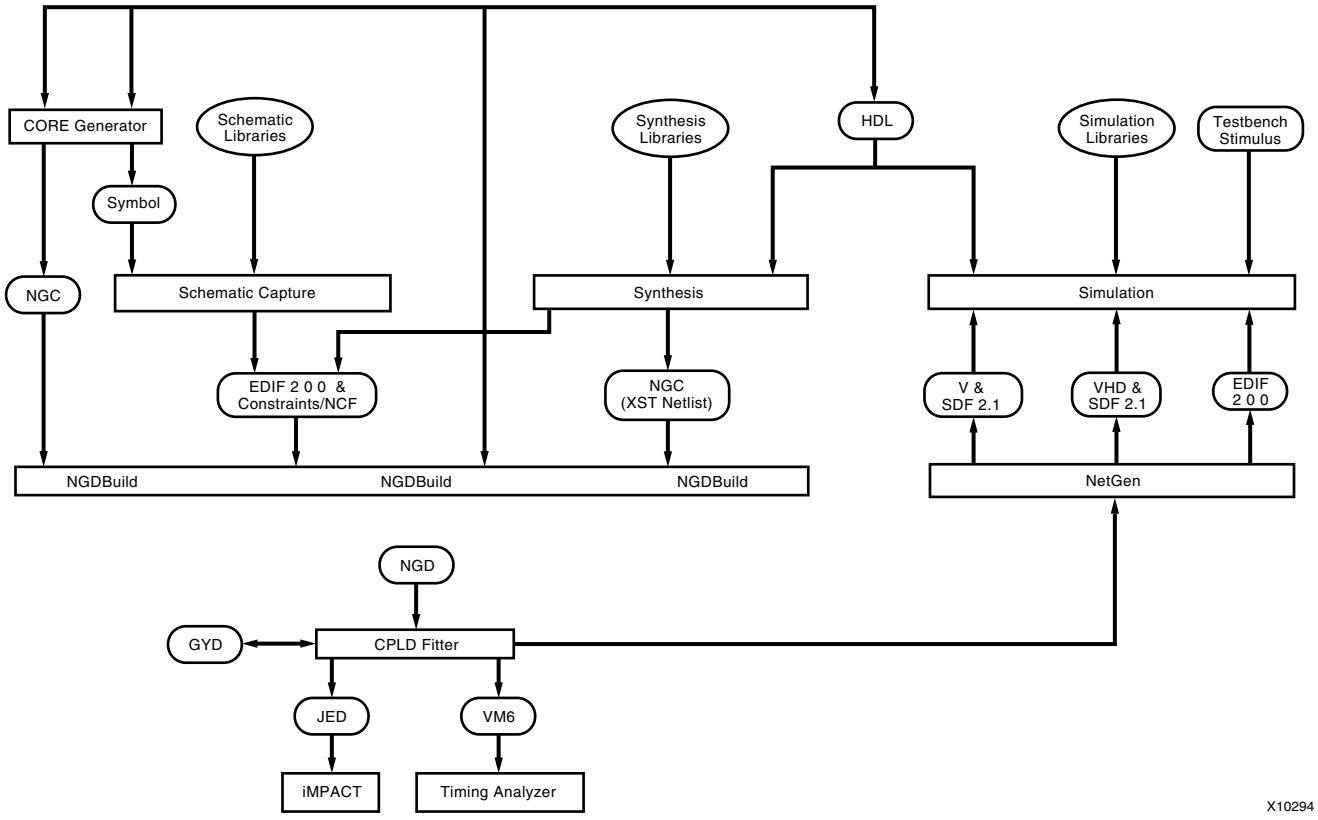


Figure 2-3: **Xilinx Software Design Flow (CPLDs)**

Design Entry and Synthesis

You can enter a design with a schematic editor or a text-based tool. Design entry begins with a design concept, expressed as a drawing or functional description. From the original design, a netlist is created, then synthesized and translated into a native generic object (NGO) file. This file is fed into the Xilinx software program called NGDBuild, which produces a logical native generic database (NGD) file.

The following figure shows the design entry and synthesis process.

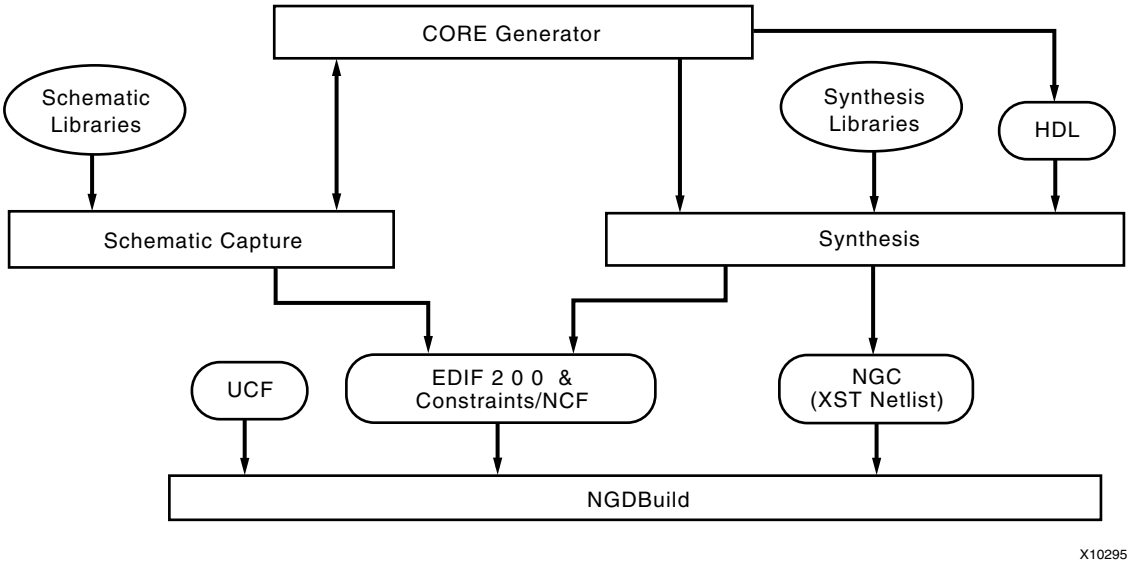


Figure 2-4: **Design Entry Flow**

Hierarchical Design

Design hierarchy is important in both schematic and HDL entry for the following reasons:

- Helps you conceptualize your design
- Adds structure to your design
- Promotes easier design debugging
- Makes it easier to combine different design entry methods (schematic, HDL, or state editor) for different parts of your design
- Makes it easier to design incrementally, which consists of designing, implementing, and verifying individual parts of a design in stages
- Reduces optimization time
- Facilitates concurrent design, which is the process of dividing a design among a number of people who develop different parts of the design in parallel.

In hierarchical designing, a specific hierarchical name identifies each library element, unique block, and instance you create. The following example shows a hierarchical name with a 2-input OR gate in the first instance of a multiplexer in a 4-bit counter:

```
/Acc/alu_1/mult_4/8count_3/4bit_0/mux_1/or2
```

Xilinx strongly recommends that you name the components and nets in your design. These names are preserved and used by the FPGA Editor tool. These names are also used for back-annotation and appear in the debug and analysis tools. If you do not name your components and nets, the schematic editor automatically generates the names. For example, if left unnamed, the software might name the previous example, as follows:

```
/$1a123/$1b942/$1c23/$1d235/$1e121/$1g123/$1h57
```

Note: It is difficult to analyze circuits with automatically generated names, because the names only have meaning for Xilinx software.

Schematic Entry Overview

Schematic tools provide a graphic interface for design entry. You can use these tools to connect symbols representing the logic components in your design. You can build your design with individual gates, or you can combine gates to create functional blocks. This section focuses on ways to enter functional blocks using library elements and the CORE Generator.

Library Elements

Primitives and macros are the "building blocks" of component libraries. Xilinx libraries provide primitives, as well as common high-level macro functions. Primitives are basic circuit elements, such as AND and OR gates. Each primitive has a unique library name, symbol, and description. Macros contain multiple library elements, which can include primitives and other macros.

You can use the following types of macros with Xilinx FPGAs:

- Soft macros have pre-defined functionality but have flexible mapping, placement, and routing. Soft macros are available for all FPGAs.
- Relationally placed macros (RPMs) have fixed mapping and relative placement. RPMs are available for all device families, except the XC9500 family.

Macros are not available for synthesis because synthesis tools have their own module generators and do not require RPMs. If you wish to override the module generation, you can instantiate CORE Generator modules. For most leading-edge synthesis tools, this does not offer an advantage unless it is for a module that cannot be inferred.

CORE Generator Tool (FPGAs Only)

The Xilinx CORE Generator design tool delivers parameterizable cores that are optimized for Xilinx FPGAs. The library includes cores ranging from simple delay elements to complex DSP (Digital Signal Processing) filters and multiplexers. For details, refer to the *CORE Generator Guide*. You can also refer to the Xilinx IP (Intellectual Property) Center Web site at http://www.xilinx.com/ipcenter, which offers the latest IP solutions. These solutions include design reuse tools, free reference designs, DSP and PCI solutions, IP implementation tools, cores, specialized system level services, and vertical application IP solutions.

HDL Entry and Synthesis

A typical Hardware Description Language (HDL) supports a mixed-level description in which gate and netlist constructs are used with functional descriptions. This mixed-level capability enables you to describe system architectures at a high level of abstraction, then incrementally refine the detailed gate-level implementation of a design.

HDL descriptions offer the following advantages:

- You can verify design functionality early in the design process. A design written as an HDL description can be simulated immediately. Design simulation at this high level, at the gate-level before implementation, allows you to evaluate architectural and design decisions.
- An HDL description is more easily read and understood than a netlist or schematic description. HDL descriptions provide technology-independent documentation of a design and its functionality. Because the initial HDL design description is technology independent, you can use it again to generate the design in a different technology, without having to translate it from the original technology.
- Large designs are easier to handle with HDL tools than schematic tools.

After you create your HDL design, you must synthesize it. During synthesis, behavioral information in the HDL file is translated into a structural netlist, and the design is optimized for a Xilinx device. Xilinx supports HDL synthesis tools for several third-party synthesis vendors. In addition, Xilinx offers its own synthesis tool, Xilinx Synthesis Technology (XST). See the *Xilinx Synthesis Technology (XST) User Guide* for information. For detailed information on synthesis, see the *Synthesis and Simulation Design Guide*.

Functional Simulation

After you create your design, you can simulate it. Functional simulation tests the logic in your design to determine if it works properly. You can save time during subsequent design steps if you perform functional simulation early in the design flow. See "Simulation" for more information.

Constraints

You may want to constrain your design within certain timing or placement parameters. You can specify mapping, block placement, and timing specifications.

You can enter constraints manually or use the Constraints Editor, Floorplanner, or FPGA Editor, which are graphical user interface (GUI) tools provided by Xilinx. You can use the Timing Analyzer GUI or TRACE command line program to evaluate the circuit against these constraints by generating a static timing analysis of your design. See Chapter 12, "TRACE" and the online Help provided with each GUI for information. See the *Constraints Guide* for detailed information on constraints.

Mapping Constraints (FPGAs Only)

You can specify how a block of logic is mapped into CLBs using an FMAP for all Spartan FPGA and Virtex FPGA families. These mapping symbols can be used in your schematic. However, if you overuse these specifications, it may be difficult to route your design.

Block Placement

Block placement can be constrained to a specific location, to one of multiple locations, or to a location range. Locations can be specified in the schematic, with synthesis tools, or in the User Constraints File (UCF). Poor block placement can adversely affect both the placement and the routing of a design. Only I/O blocks require placement to meet external pin requirements.

Timing Specifications

You can specify timing requirements for paths in your design. PAR uses these timing specifications to achieve optimum performance when placing and routing your design.

Netlist Translation Programs

Two netlist translation programs allow you to read netlists into the Xilinx software tools. EDIF2NGD allows you to read an Electronic Data Interchange Format (EDIF) 2 0 0 file. The NGDBuild program automatically invokes these programs as needed to convert your EDIF file to an NGD file, the required format for the Xilinx software tools. NGC files output from the Xilinx XST synthesis tool are read in by NGDBuild directly.

You can find detailed descriptions of the EDIF2NGD, and NGDBuild programs in Chapter 6, "NGDBuild" and "Appendix B".

Design Implementation

Design Implementation begins with the mapping or fitting of a logical design file to a specific device and is complete when the physical design is successfully routed and a bitstream is generated. You can alter constraints during implementation just as you did during the Design Entry step. See "Constraints" for information.

The following figure shows the design implementation process for FPGA designs:

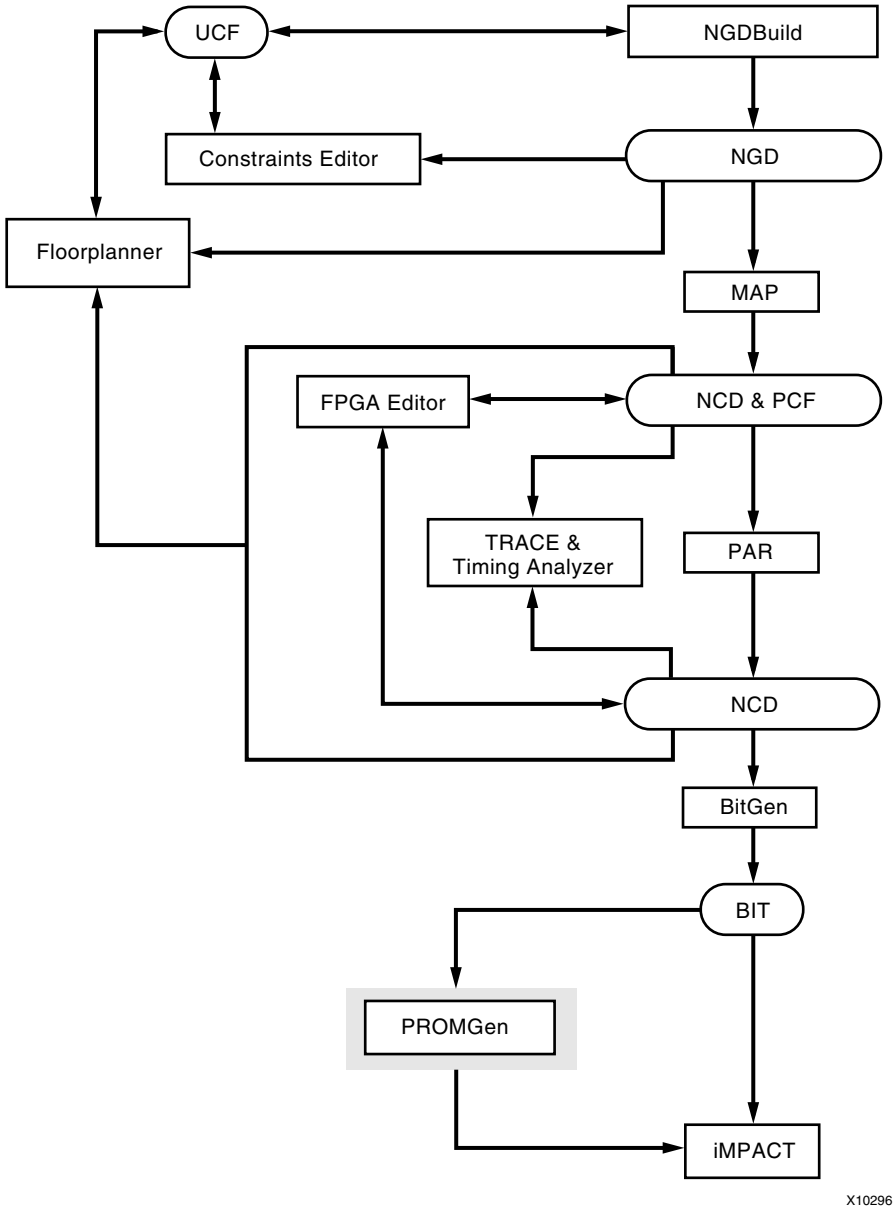


Figure 2-5: **Design Implementation Flow (FPGAs)**

The following figure shows the design implementation process for CPLD designs:

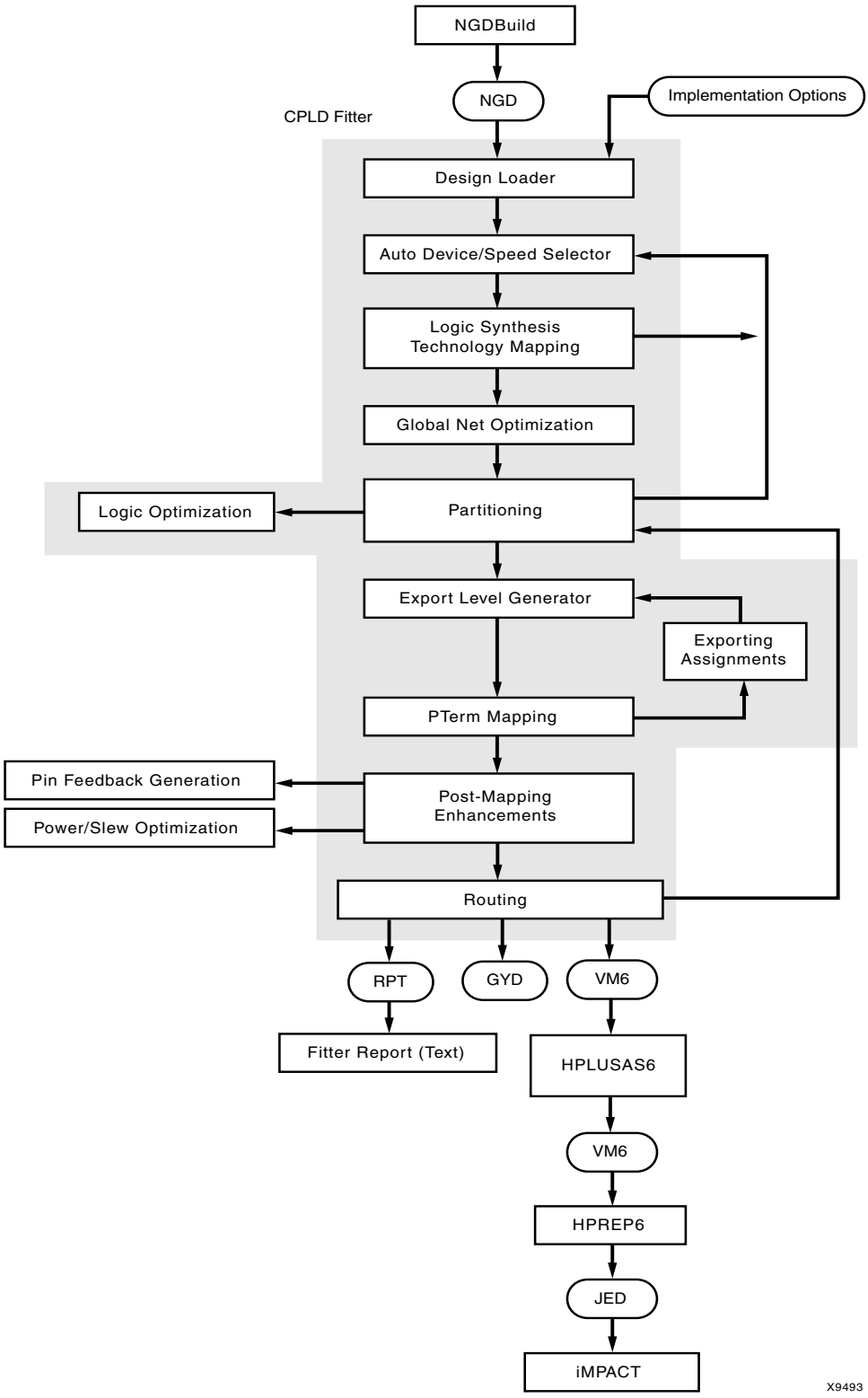


Figure 2-6: **Design Implementation Flow (CPLDs)**

Mapping (FPGAs Only)

For FPGAs, the MAP command line program maps a logical design to a Xilinx FPGA. The input to MAP is an NGD file, which contains a logical description of the design in terms of both the hierarchical components used to develop the design and the lower-level Xilinx primitives, and any number of NMC (hard placed-and-routed macro) files, each of which contains the definition of a physical macro. MAP then maps the logic to the components (logic cells, I/O cells, and other components) in the target Xilinx FPGA.

The output design from MAP is an NCD file, which is a physical representation of the design mapped to the components in the Xilinx FPGA. The NCD file can then be placed and routed, using the PAR command line program. See Chapter 7, "MAP" for detailed information.

Placing and Routing (FPGAs Only)

For FPGAs, the PAR command line program takes a mapped NCD file as input, places and routes the design, and outputs a placed and routed NCD file, which is used by the bitstream generator, BitGen. The output NCD file can also act as a guide file when you reiterate placement and routing for a design to which minor changes have been made after the previous iteration. See Chapter 9, "PAR" for detailed information.

You can also use the FPGA Editor GUI tool to do the following:

- Place and route critical components before running automatic place and route tools on an entire design
- Modify placement and routing manually; the editor allows both automatic and manual component placement and routing

Note: For more information, see the online Help provided with the FPGA Editor.

Bitstream Generation (FPGAs Only)

For FPGAs, the BitGen command line program produces a bitstream for Xilinx device configuration. BitGen takes a fully routed NCD file as its input and produces a configuration bitstream—a binary file with a .bit extension. The BIT file contains all of the configuration information from the NCD file defining the internal logic and interconnections of the FPGA, plus device-specific information from other files associated with the target device. See Chapter 14, "BitGen" for detailed information.

After you generate your BIT file, you can download it to a device using the iMPACT GUI. You can also format the BIT file into a PROM file using the PromGen command line program and then download it to a device using the iMPACT GUI. See Chapter 16, "PROMGen" of this guide or the *iMPACT online help* for more information.

Design Verification

Design verification is testing the functionality and performance of your design. You can verify Xilinx designs in the following ways:

- Simulation (functional and timing)
- Static timing analysis
- In-circuit verification

The following table lists the different design tools used for each verification type.

Table 2-1: **Verification Tools**

Verification Type	Tools
Simulation	Third-party simulators (integrated and non-integrated)
Static Timing Analysis	TRACE (command line program) Timing Analyzer (GUI) Mentor Graphics® TAU and Innoveda BLAST software for use with the STAMP file format (for I/O timing verification only)
In-Circuit Verification	Design Rule Checker (command line program) Download cable

Design verification procedures should occur throughout your design process, as shown in the following figures.

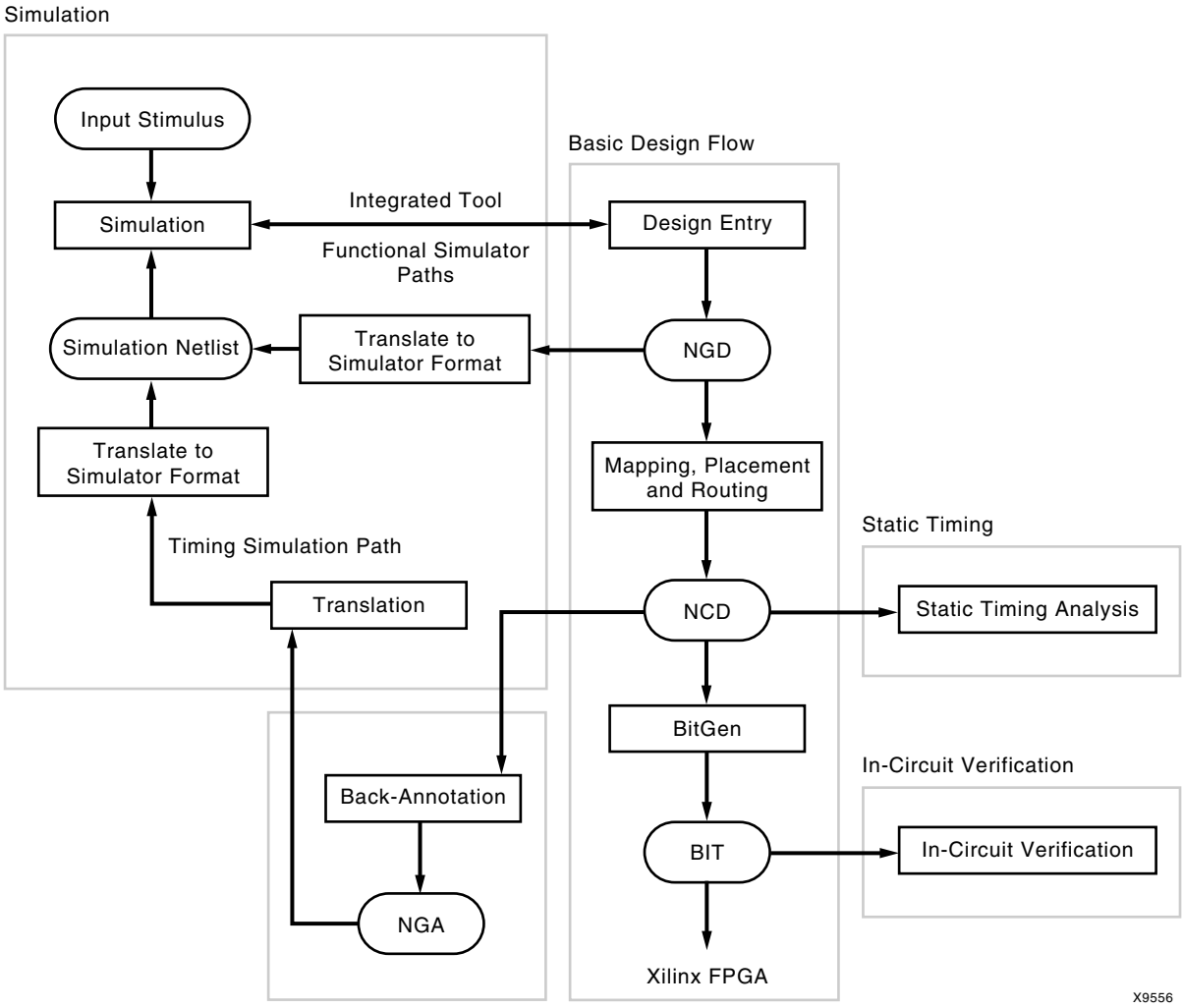


Figure 2-7: **Three Verification Methods of the Design Flow (FPGAs)**

The following figure shows the verification methods of the design flow for CPLDs.

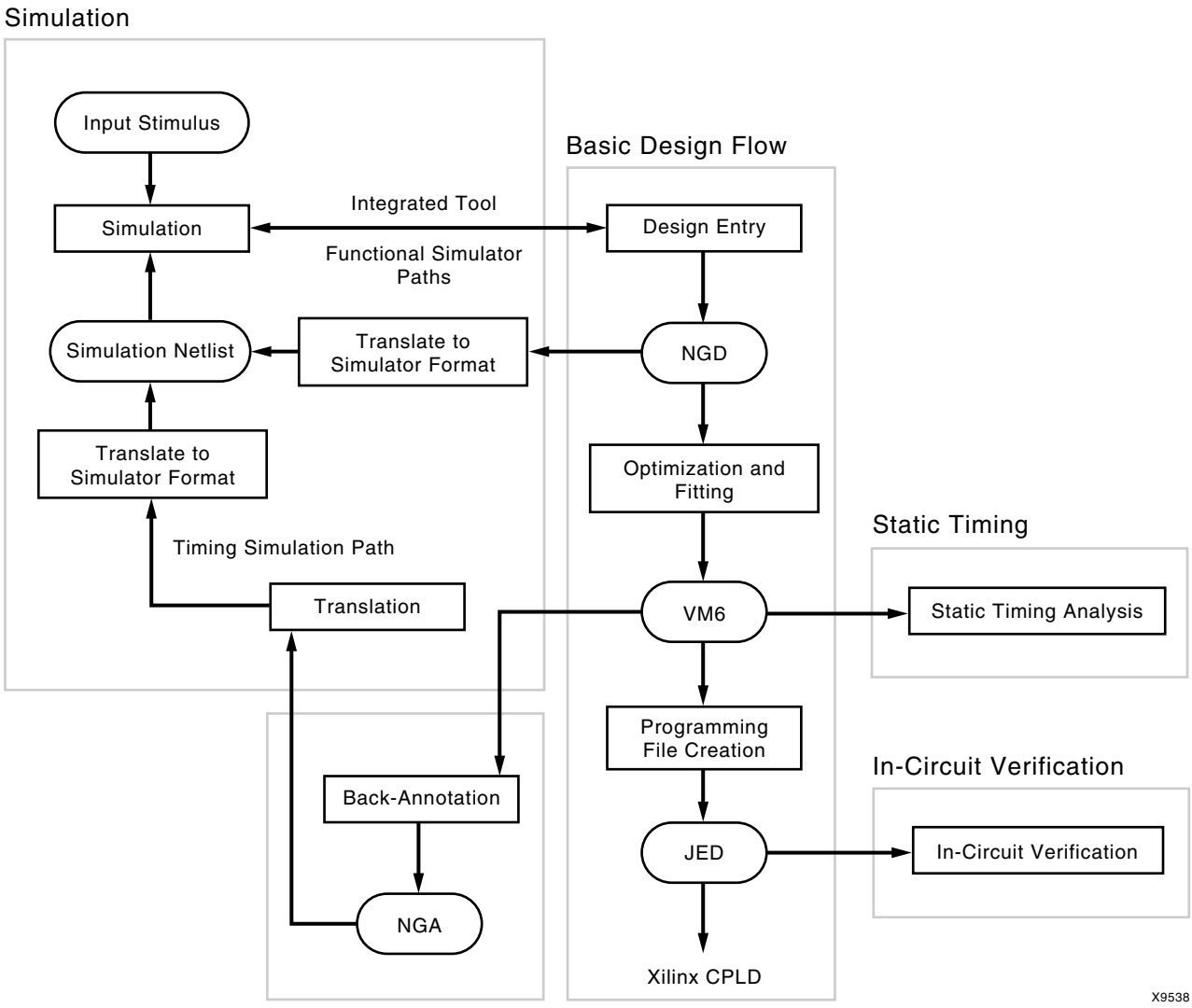


Figure 2-8: **Three Verification Methods of the Design Flow (CPLDs)**

Simulation

You can run functional or timing simulation to verify your design. This section describes the back-annotation process that must occur prior to timing simulation. It also describes the functional and timing simulation methods for both schematic and HDL-based designs.

Back-Annotation

Before timing simulation can occur, the physical design information must be translated and distributed back to the logical design. For FPGAs, this back-annotation process is done with a program called NetGen. For CPLDs, back-annotation is performed with the TSim Timing Simulator. These programs create a database, which translates the back-annotated information into a netlist format that can be used for timing simulation.

The following figures show the back-annotation flows:

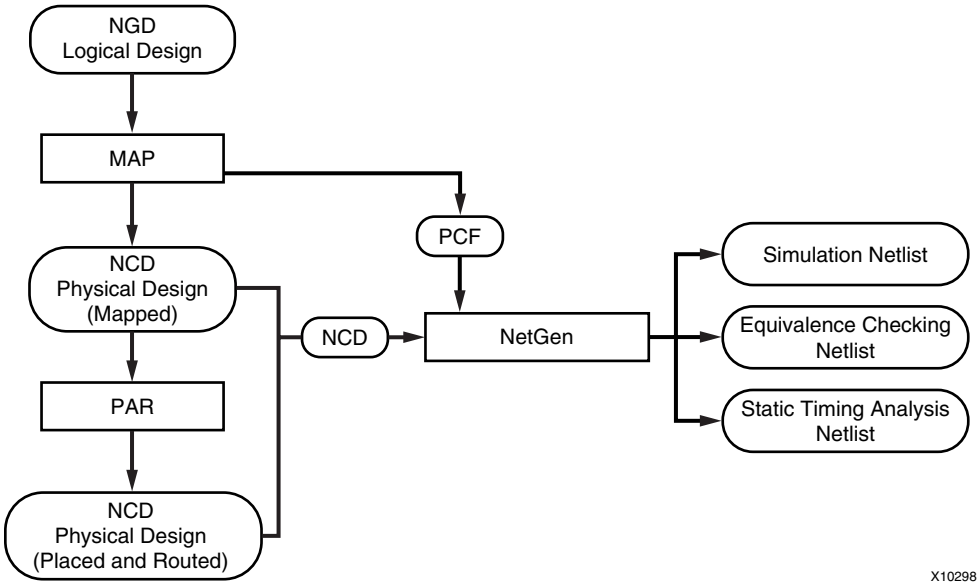


Figure 2-9: **Back-Annotation Flow for FPGAs**

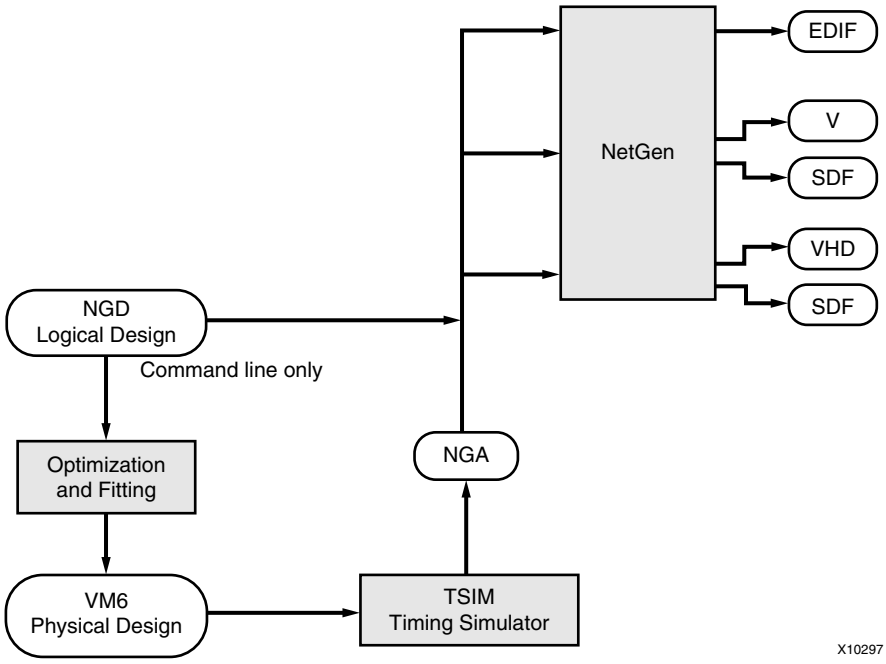


Figure 2-10: **Back-Annotation (CPLDs)**

NetGen

NetGen is a command line program that distributes information about delays, setup and hold times, clock to out, and pulse widths found in the physical NCD design file back to the logical NGD file and generates a Verilog or VHDL netlist for use with supported timing simulation, equivalence checking, and static timing analysis tools.

NetGen reads an NCD as input. The NCD file can be a mapped-only design, or a partially or fully placed and routed design. An NGM file, created by MAP, is an optional source of input. NetGen merges mapping information from the optional NGM file with placement, routing, and timing information from the NCD file.

Note: NetGen reads an NGA file as input to generate a timing simulation netlist for CPLD designs.

See Chapter 22, "NetGen" for detailed information.

Schematic-Based Simulation

Design simulation involves testing your design using software models. It is most effective when testing the functionality of your design and its performance under worst-case conditions. You can easily probe internal nodes to check the behavior of your circuit, and then use these results to make changes in your schematic.

Simulation is performed using third-party tools that are linked to the Xilinx Development System. Use the various CAE-specific interface user guides, which cover the commands and features of the Xilinx-supported simulators, as your primary reference.

The software models provided for your simulation tools are designed to perform detailed characterization of your design. You can perform functional or timing simulation, as described in the following sections.

Functional Simulation

Functional simulation determines if the logic in your design is correct before you implement it in a device. Functional simulation can take place at the earliest stages of the design flow. Because timing information for the implemented design is not available at this stage, the simulator tests the logic in the design using unit delays.

Note: It is usually faster and easier to correct design errors if you perform functional simulation early in the design flow.

You can use integrated and non-integrated simulation tools. Integrated tools, such as Mentor Graphics or Innoveda, often contain a built-in interface that links the simulator and a schematic editor, allowing the tools to use the same netlist. You can move directly from entry to simulation when using a set of integrated tools.

Functional simulation in schematic-based tools is performed immediately after design entry in the capture environment. The schematic capture tool requires a Xilinx Unified Library and the simulator requires a library if the tools are not integrated. Most of the schematic-based tools require translation from their native database to EDIF for implementation. The return path from implementation is usually EDIF with certain exceptions in which a schematic tool is tied to an HDL simulator.

Timing Simulation

Timing simulation verifies that your design runs at the desired speed for your device under worst-case conditions. This process is performed after your design is mapped, placed, and routed for FPGAs or fitted for CPLDs. At this time, all design delays are known.

Timing simulation is valuable because it can verify timing relationships and determine the critical paths for the design under worst-case conditions. It can also determine whether or not the design contains set-up or hold violations.

Before you can simulate your design, you must go through the back-annotation process, as described in "Back-Annotation". During this process, NetGen creates suitable formats for various simulators.

Note: Naming the nets during your design entry is important for both functional and timing simulation. This allows you to find the nets in the simulations more easily than looking for a software-generated name.

HDL-Based Simulation

Xilinx supports functional and timing simulation of HDL designs at the following points:

- Register Transfer Level (RTL) simulation, which may include the following:
  - Instantiated UniSim library components
  - LogiCORE models
- Post-synthesis functional simulation with one of the following:
  - Gate-level UniSim library components
  - Gate-level pre-route SimPrim library components
- Post-implementation back-annotated timing simulation with the following:
  - SimPrim library components
  - Standard delay format (SDF) file

The following figure shows when you can perform functional and timing simulation:

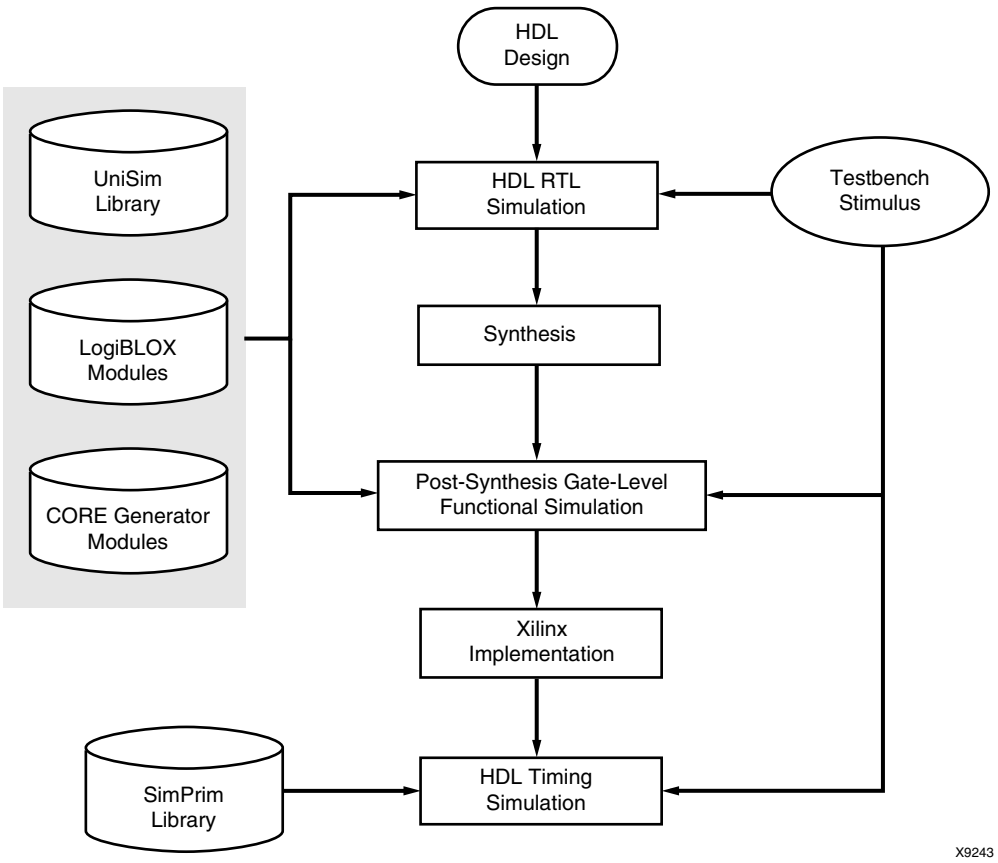


Figure 2-11: **Simulation Points for HDL Designs**

The three primary simulation points can be expanded to allow for two post-synthesis simulations. These points can be used if the synthesis tool cannot write VHDL or Verilog, or if the netlist is not in terms of UniSim components. The following table lists all the simulation points available in the HDL design flow.

Table 2-2: **Five Simulation Points in HDL Design Flow**

Simulation	UniSim	SimPrim	SDF
RTL	X		
Post-Synthesis	X		
Functional Post-NGDBuild (Optional)		X	
Functional Post-MAP (Optional)		X	X
Post-Route Timing		X	X

These simulation points are described in the "Simulation Points" section of the *Synthesis and Simulation Design Guide*.

The libraries required to support the simulation flows are described in detail in the "VHDL/Verilog Libraries and Models" section of the *Synthesis and Simulation Design Guide*. The flows and libraries support close functional equivalence of initialization behavior between functional and timing simulations. This is due to the addition of new methodologies and library cells to simulate Global Set/Reset (GSR) and Global 3-State (GTS) behavior.

You must address the built-in reset circuitry behavior in your designs, starting with the first simulation, to ensure that the simulations agree at the three primary points. If you do not simulate GSR behavior prior to synthesis and place and route, your RTL and post-synthesis simulations may not initialize to the same state as your post-route timing simulation. If this occurs, your various design descriptions are not functionally equivalent and your simulation results do not match.

In addition to the behavioral representation for GSR, you must add a Xilinx implementation directive. This directive is specifies to the place and route tools to use the special purpose GSR net that is pre-routed on the chip, and not to use the local asynchronous set/reset pins. Some synthesis tools can identify the GSR net from the behavioral description, and place the STARTUP module on the net to direct the implementation tools to use the global network. However, other synthesis tools interpret behavioral descriptions literally and introduce additional logic into your design to implement a function. Without specific instructions to use device global networks, the Xilinx implementation tools use general-purpose logic and interconnect resources to redundantly build functions already provided by the silicon.

Even if GSR behavior is not described, the chip initializes during configuration, and the post-route netlist has a net that must be driven during simulation. The "Understanding the Global Signals for Simulation" section of the *Synthesis and Simulation Design Guide* includes the methodology to describe this behavior, as well as the GTS behavior for output buffers.

Xilinx VHDL simulation supports the VITAL standard. This standard allows you to simulate with any VITAL-compliant simulator. Built-in Verilog support allows you to simulate with the Cadence Verilog-XL and other compatible simulators. Xilinx HDL simulation supports all current Xilinx FPGA and CPLD devices. Refer to the *Synthesis and Simulation Design Guide* for the list of supported VHDL and Verilog standards.

Static Timing Analysis (FPGAs Only)

Static timing analysis is best for quick timing checks of a design after it is placed and routed. It also allows you to determine path delays in your design. Following are the two major goals of static timing analysis:

- Timing verification

  This is verifying that the design meets your timing constraints.

- Reporting

  This is enumerating input constraint violations and placing them into an accessible file. You can analyze partially or completely placed and routed designs. The timing information depends on the placement and routing of the input design.

You can run static timing analysis using the Timing Reporter and Circuit Evaluator (TRACE) command line program. See Chapter 12, "TRACE" for detailed information. You can also use the Timing Analyzer GUI to perform this function. See the online Help provided with the Timing Analyzer for additional information. Use either tool to evaluate how well the place and route tools met the input timing constraints.

In-Circuit Verification

As a final test, you can verify how your design performs in the target application. In-circuit verification tests the circuit under typical operating conditions. Because you can program your Xilinx devices repeatedly, you can easily load different iterations of your design into your device and test it in-circuit. To verify your design in-circuit, download your design bitstream into a device with the Parallel Cable IV or MultiPRO cable.

Note: For information about Xilinx cables and hardware, see the *iMPACT online help.*

Design Rule Checker (FPGAs Only)

Before generating the final bitstream, it is important to use the DRC option in BitGen to evaluate the NCD file for problems that could prevent the design from functioning properly. DRC is invoked automatically unless you use the –d option. See Chapter 8, "Physical Design Rule Check" and Chapter 14, "BitGen" and for detailed information.

Xilinx Design Download Cables

Xilinx provides the Parallel Cable IV or MultiPRO cable to download the configuration data containing the device design.

You can use the Xilinx download cables with the iMPACT Programming software for FPGA and CPLD design download and readback, and configuration data verification. The iMPACT Programming software cannot be used to perform real-time design functional verification.

Probe

The Xilinx PROBE function in FPGA Editor provides real-time debug capability good for analyzing a few signals at a time. Using PROBE a designer can quickly identify and route any internal signals to available I/O pins without having to replace and route the design. The real-time activity of the signal can then be monitored using normal lab test equipment such as logic/state analyzers and oscilloscopes.

ChipScope ILA and ChipScope PRO

The ChipScope toolset was developed to assist engineers working at the PCB level. ChipScope ILA actually embeds logic analyzer cores into your design. These logic cores allow the user to view all the internal signals and nodes within an FPGA. ChipScope ILA supports user selectable data channels from 1 to 256. The depth of the sample buffer ranges from 256 to 16384 in Virtex-II devices. Triggers are changeable in real-0time without affecting the user logic or requiring recompilation of the user design.

FPGA Design Tips

The Xilinx FPGA architecture is best suited for synchronous design. Strict synchronous design ensures that all registers are driven from the same time base with no clock skew. This section describes several tips for producing high-performance synchronous designs.

Design Size and Performance

Information about design size and performance can help you to optimize your design. When you place and route your design, the resulting report files list the number of CLBs, IOBs, and other device resources available. A first pass estimate can be obtained by processing the design through the MAP program.

If you want to determine the design size and performance without running automatic implementation software, you can quickly obtain an estimate from a rough calculation based on the Xilinx FPGA architecture.

Global Clock Distribution

Xilinx clock networks guarantee small clock skew values. The following table lists the resources available for the Xilinx FPGA families.

Table 2-3: **Global Clock Resources**

FPGA Family	Resource	Number	Destination Pins
Spartan	BUFGS	4	Clock, control, or certain input
Virtex, Virtex-E, Spartan-II, Spartan-IIE	BUFG	4	Clock
Virtex-II, Virtex-II Pro	BUFGMUX	16	Clock

Note: In certain devices families, global clock buffers are connected to control pin and logic inputs. If a design requires extensive routing, there may be extra routing delay to these loads.

Data Feedback and Clock Enable

The following figure shows a gated clock. The gated clock's corresponding timing diagram shows that this implementation can lead to clock glitches, which can cause the flip-flop to clock at the wrong time.

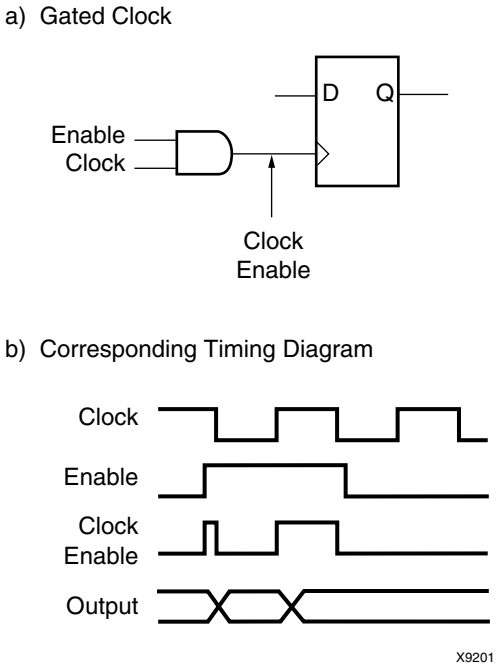


Figure 2-12: **Gated Clock**

The following figure shows a synchronous alternative to the gated clock using a data path. The flip-flop is clocked at every clock cycle and the data path is controlled by an enable. When the enable is Low, the multiplexer feeds the output of the register back on itself. When the enable is High, new data is fed to the flip-flop and the register changes its state.

This circuit guarantees a minimum clock pulse width and it does not add skew to the clock. The Spartan-II, and Virtex families' flip-flops have a built-in clock-enable (CE).

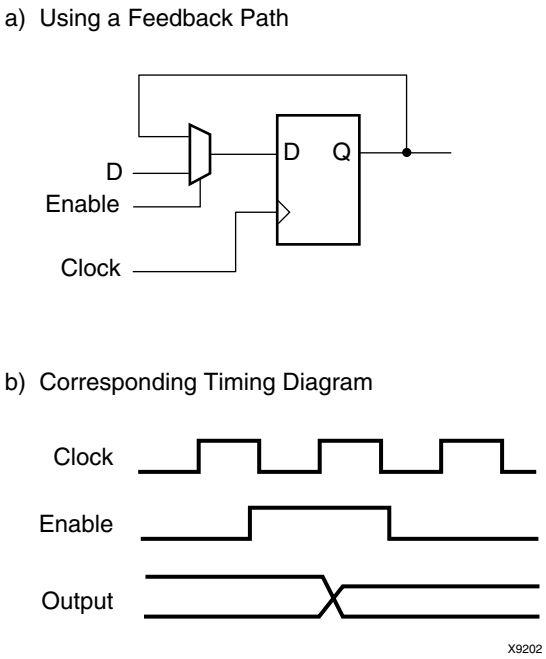


Figure 2-13: **Synchronous Design Using Data Feedback**

Counters

Cascading several small counters to create a larger counter is similar to a gated clock. For example, if two 8-bit counters are connected, the terminal counter (TC) of the first counter is a large AND function gating the second clock input.

The following figure shows how you can create a synchronous design using the CE input. In this case, the TC of the first stage is connected directly to the CE of the second stage.

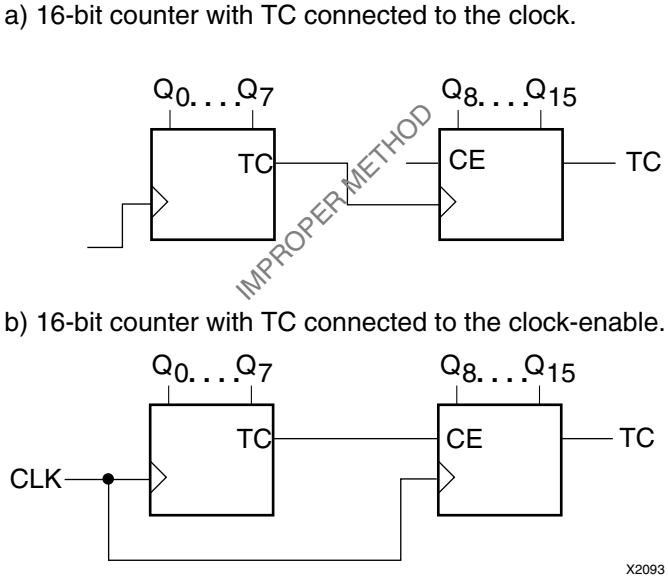


Figure 2-14: **Two 8-Bit Counters Connected to Create a 16-Bit Counter**

Other Synchronous Design Considerations

Other considerations for achieving a synchronous design include the following:

- Use clock enables instead of gated clocks to control the latching of data into registers.
- If your design has more clocks than the number of global clock distribution networks, try to redesign to reduce the number of clocks. Otherwise, put the clocks that have the lowest fanout onto normally routed nets, and specify a low MAXSKEW rating. A clock net routed through a normal net has skew.
- Use the Virtex low skew resources. Make sure the MAXSKEW rating is *not* specified when using these resources.

Chapter 3

Tcl

Xilinx Tcl commands are compatible with the following device families:

- Virtex™, Virtex™-E
- Virtex™-II
- Virtex™-II Pro, Virtex™-II Pro X
- Virtex™-4
- Virtex ™-5 LX
- Spartan™-II, Spartan™-IIE
- Spartan™-3, Spartan™-3E, Spartan™-3L

This chapter describes the Xilinx Tcl Shell (xtclsh) and the Xilinx Tcl commands. This chapter contains the following sections:

Tcl Overview

Tool Command Language (Tcl) is an easy to use scripting language and an industry standard popular in the electronic design automation (EDA) industry.

The Xilinx Tcl command language is designed to complement and extend the graphical user interface (GUI). For new users and projects, the GUI provides an easy-to-use interface to set up a project, perform initial implementations, explore available options, set constraints, and visualize the design. Alternatively, for users that know exactly what options and implementation steps they wish to perform, the Xilinx Tcl commands provide a batch interface that makes it convenient to execute the exact same script or steps over and over again. By making the syntax of the Xilinx Tcl commands match the GUI interaction as closely as possible, Xilinx Tcl commands make it easy to transition from using the GUI to running the tools in script or batch mode. A list of available project properties and batch tool options can be found in the "Project Properties and Options" section of this chapter.

The Xilinx Tcl command language also supports more advanced scripting techniques. Xilinx Tcl commands provide support for collections and objects that allow advanced users to write scripts that query the design and implementation, and then to take appropriate actions based on the results. "Tcl Commands for Advanced Scripting" are described in more detail later in the this chapter.

Tcl commands are accessed from the Xilinx Tcl Shell (xtclsh), which is available from the command line, or from the Tcl Console tab in Project Navigator. Xilinx Tcl commands are categorized in two ways: general usage and advanced scripting. See the "Xilinx Tcl Commands" section of this chapter for a detailed listing that includes a description, syntax, an example, and the Tcl return for each command.

Xilinx Tcl Shell

To access the Xilinx Tcl Shell (xtclsh) from Project Navigator, click the Tcl Console tab, which displays a window with the xtclsh prompt (%).

To access the xtclsh from the command line, type **`xtclsh`** from the command prompt to return the xtclsh prompt (%). Example:

```
> xtclsh
%
```

Command line syntax is based on the Tcl command and corresponding subcommand that you enter. For example:

```
% <tcl_command> <subcommand> <optional_arguments>
```

tcl_command is the name of the Xilinx Tcl command.

subcommand is the subcommand name for the Xilinx Tcl command.

optional_arguments are the arguments specific to each subcommand. Example syntax for all Xilinx Tcl commands, subcommands, and their respective arguments is included in the "Tcl Commands for General Usage" and "Tcl Commands for Advanced Scripting" sections of this chapter.

Accessing Help

Use the *help* command to get detailed information on Xilinx-specific Tcl commands. From the xtclsh prompt (%), type **`help`** for a list and brief description of Xilinx Tcl commands. For help on a specific Tcl command, type the following:

```
% help <tcl_command>
```

You can also get information on a specific subcommand by typing the subcommand name after the Tcl command. For example, type the following to get help on creating a new ISE project:

```
% help project new
```

help is the command that calls the Tcl help information.

project specifies the name of the Xilinx Tcl command.

new specifies the name of the project subcommand you wish to obtain help information on.

In Project Navigator, Tcl help is accessed from the Tcl Console tab using the same syntax as described above.

Tcl Fundamentals

This section provides some very basic information about the syntactic style of Xilinx Tcl commands. For more information about Tcl in general, please refer to Tcl documentation easily available on the internet, for example: http://www.tck.tk/doc , which is the website for the Tcl Developer Xchange.

In general, Tcl commands are procedural. Each Tcl command is a series of words, with the first word being the command name. For Xilinx Tcl commands, the command name is either a noun (e.g., project) or a verb (e.g., search). For commands that are nouns, the second word on the command line is the verb (e.g., project open). This second word is called the subcommand.

Subsequent words on the command line are additional parameters to the command. For Xilinx Tcl commands, required parameters are positional, which means they must always be specified in an exact order and follow the subcommand. Optional parameters follow the required parameters, can be specified in any order, and always have a flag that starts with "-"to indicate the parameter name; for example, -instance <*instance-name*>.

Tcl is case sensitive. Xilinx Tcl command names are always lower case. If the name is two words, the words are joined with an underscore (_). Even though Tcl is case sensitive, most design data (e.g., an instance name), property names, and property values are case insensitive. To make it less burdensome to type at the command prompt, unique prefixes are recognized when typing a subcommand, which means only typing the first few letters of a command name is all that is required for it to be recognized. Unique prefixes are also recognized for partition properties and property values.

The real power of Tcl is unleashed when it is used for nested commands and for scripting. The result of any command can be stored in a variable. Values are assigned to variables and properties with the set command. The set command takes two arguments. The first argument is the name of the variable and the second argument is the value. It is not necessary to declare Tcl variables before you use them. If one does not exist, it is created when the command is executed.

In Tcl, the dollar-sign ($) syntax is used to substitute a variable's value for its name. For example, $foo in a Tcl command is replaced by the value of the variable foo.

The result of a command can also be substituted directly into another command. Tcl uses square brackets [ ] for these nested commands. Tcl interprets everything between square brackets [ ] as a command and substitutes the command result for the text within the square brackets [ ].

Tcl provides several ways to quote strings that contain spaces or other special characters and to manage substitution. Double quotes (") allow some special characters ([ ] and $) for substitution. Curly braces { } perform no substitutions.

For very specific command line examples, please see the "Tcl Commands for General Usage" and "Tcl Commands for Advanced Scripting" sections of this chapter.

Xilinx Namespace

All Xilinx Tcl commands are part of the Tcl namespace *::xilinx::*. If another Tcl package uses a command name that conflicts with a Xilinx-specific Tcl command name, the Xilinx namespace must be used to access the command. For example, type the following to create a new project using Xilinx-specific Tcl commands:

```
% xilinx::project new <project_name>
```

It is only necessary to specify the Xilinx namespace when you have more than one namespace installed.

Xilinx Tcl Commands

The following sections include detailed listings of Xilinx Tcl commands, which are divided into two parts:

- Tcl Commands for General Usage
- Tcl Commands for Advanced Scripting

Each detailed listing includes the description, syntax, an example, and the Tcl return for each command.

In most cases, the examples shown assume that a project has been created with the *project new* command or a project has been opened with the *project open* command. Project files are added with the *xfile add* command.

To view how Xilinx Tcl commands can be used in a realistic way, see the "Example Tcl Scripts" located at the end of this chapter.

The following tables summarize the Xilinx Tcl commands based on those for general usage and those for advanced scripting.

Table 3-1: **Xilinx Tcl Commands for General Usage**

Commands	Subcommands
partition (support design preservation)	delete get new properties rerun set
process (run and manage project processes)	run

Table 3-1: **Xilinx Tcl Commands for General Usage**

Commands	Subcommands
project (create and manage projects)	clean close delete get get_processes new open properties set
timing_analysis (generate timing analysis reports)	delete disable_constraints disable_cpt enable_constraints enable_cpt get new reset run saveas set set_constraint set_endpoints set_filter set_query show_settings
xfile (manage project files)	add get remove

Table 3-2: **Xilinx Tcl Commands for Advanced Scripting**

Commands	Subcommands
collection (create and manage a collection)	append_to copy equal foreach get index properties remove_from set sizeof
object (get object information)	get name properties type
search (search and return matching objects)	

Tcl Commands for General Usage

This section describes the Xilinx Tcl commands for general usage. To view a sample script of how these commands are used, see the "Sample Tcl Script for General Usage" at the end of this chapter.

partition (support design preservation)

The *partition* command is used to create and manage partitions, which are used for design preservation. A Partition is set on an instance in a design. The Partition indicates that the implementation for the instance should be preserved and reused when possible.

```
% partition <subcommand>
```

Partition names should follow the naming conventions of the HDL source. In general, the local name of a top-level partition is set to the name of the design, which is based on the top-level entity or module in the HDL source. The full hierarchical name of a top-level partition is a forward slash (/) followed by the local name. For example, if the design name is stopwatch, then the hierarchical name for the top-level instance is /stopwatch. It is always necessary to give the full name of any instance or partition that you specify on the command line. Partition names are case-sensitive.

delete (delete a partition)

The *partition delete* command deletes the specified partition from the current ISE project. This command also deletes any properties on the partition; however, timing and other logic constraints on the instance are preserved.

Note: A Partition name may not be unique to the project. In this case, the full hierarchical name of the partition should be specified for the partition you wish to delete.

```
% partition delete <partition_name>
```

partition is the name of the Xilinx Tcl command.

delete is the name of the partition subcommand.

partition_name specifies the full hierarchical name of the partition you wish to remove from the project or the name of the

Example:	**% partition delete /stopwatch/Inst_dcm1**
Description:	In this example, the Inst_dcm1 partition is deleted and removed from the project repository. Note that only the partition is deleted from the project not the instance that the partition is set on.
Tcl Return:	The number of partitions deleted. In this example, 1 is returned.

get (get partition properties)

The *partition get* command returns the value of the specified partition property. Note that the preserve property is assigned with the *partition set* command.

```
% partition get <partition_name> <property_name>
```

partition is the name of the Xilinx Tcl command.

get is the name of the partition subcommand.

partition_name specifies the full hierarchical name of the partition or the collection. A collection is specified using the dollar-sign syntax ($) with the name of the collection variable.

property_name specifies the name of the property you wish to get the value of. Valid partition property names and their Tcl returns are shown in the following table:

Table 3-3: **Partition Property Names and Tcl Returns**

Partition Property Name	Tcl Return
name	The name of the partition.
parent	The name of the parent of the partition. If the partition is the top-level partition, the returned name is empty.
children	A collection of the child partitions. If the partition has no children, the returned collection is empty.
preserve	Routing, placement, synthesis, or inherit
preserve_effective	Returns the inherited value for the preserve property.

Table 3-3: **Partition Property Names and Tcl Returns**

Partition Property Name	Tcl Return
up_to_date_synthesis	True or false, based on the status of the synthesis results.
up_to_date_implementation	True or false, based on the status of the implementation results.

Example:	**% partition get /stopwatch/Inst_dcm1 preserve**
Description:	In this example, the *partition get* command is used to obtain the current value of the preserve property.
Tcl Return:	The property value as a text string. In this example, the return will be routing, placement, synthesis, or inherit.

new (create a new partition)

The *partition new* command creates a new partition on a specified instance or collection in the current design. A collection is specified using the dollar-sign syntax ($) with the name of the collection variable.

```
% partition new <partition_name>
```

partition is the name of the Xilinx Tcl command.

new is the name of the partition subcommand.

partition_name specifies the full hierarchical name of the instance you wish to create the partition on, or the collection.

Example:	**% partition new /stopwatch/Inst_dcm1**
Description:	In this example, the *partition new* command is used to create a new partition on the Inst_dcm1 instance in the current design. The full hierarchical name (/stopwatch/Inst_dcm1) is required to specify the instance. In this case, stopwatch is the name of the top-level entity in the VHDL source.
Tcl Return:	The full hierarchical name of the newly created partition. In this example, /stopwatch/Inst_dcm1 is returned.

properties (list available partition properties)

The *partition properties* command displays a list of the supported properties for all partitions. You can set the value of any property with the *partition set* command.

```
% partition properties
```

partition is the name of the Xilinx Tcl command.

properties is the name of the partition subcommand.

Example:	**% partition properties**
Description:	In this example, the *partition properties* command is used to list the properties available for all partitions in the current ISE project.
Tcl Return:	The available partition properties as a Tcl list.

For a list of partition properties and their return values, see the "set (set partition preserve property)" command in this chapter.

rerun (force partition synthesis and implementation)

The *partition rerun* command forces re-synthesis or re-implementation of a specified partition. If synthesis is specified, synthesis (XST), translation (NGDBuild), packing (MAP), and placement and routing (PAR) are all performed the next time the *process run* command is specified. If implementation is specified, translation, packing, and placement and routing are performed.

```
% partition rerun <partition_name> {synthesis|implementation}
```

partition is the name of the Xilinx Tcl command.

rerun is the name of the partition subcommand.

partition_name specifies the full hierarchical name of the partition or the collection you wish to force the re-synthesis or re-implementation of. A collection is specified using the dollar-sign syntax ($) with the name of the collection variable.

synthesis specifies re-synthesis of the partition starting with XST, then NGDBuild, MAP, and PAR.

implementation specifies re-implementing the partition starting with NGDBuild, then MAP and PAR.

Example:	**% partition rerun implementation /stopwatch/Inst_dcm1**
Description:	In this example, the *partition rerun* command forces the re-implementation of the /stopwatch/Inst_dcm1 partition.
Tcl Return:	True if the command was successful; false otherwise.

Note: This command is used with the *process run* command, which runs the processes of the project.

set (set partition preserve property)

The *partition set* command assigns the partition preserve property and value for the specified partition.

```
% partition set <partition> preserve <value>
```

partition is the name of the Xilinx Tcl command.

set is the name of the partition subcommand.

partition specifies the full hierarchical name of the partition or the collection you wish to set the property for. A collection is specified using the dollar-sign syntax ($) with the name of the collection variable.

preserve is the property used to control the level of changes that can be made to the implementation of partitions that have not been re-implemented. Values for the *preserve* property are:

```
preserve {routing|placement|synthesis|inherit}
```

routing -- Most data preservation comes from routing. When the property value is set to routing, all implementation data is preserved, including synthesis, packing, placement, and routing. Routing is the default property value.

placement --This is the second-highest property value for the preserve property. With this setting, synthesis, packing, and placement are preserved. Routing is only re-implemented if another partition requires the resources.

synthesis -- This is the lowest-level preserve property value because only the netlist, which contains synthesis information, is preserved. With this setting, packing, placement and routing are open for re-implementation; however, placement and routing are only re-implemented if another partition requires the resources.

inherit -- This value specifies that the partition inherits the same preserve property value as its parent partition. Inherit is the default setting for all child partitions. This setting is not available for top-level partitions.

Example:	**% partition set /stopwatch/Inst_dcm1 preserve synthesis**
Description:	In this example, the *partition set* command is used to specify the preserve property for the Inst_dcm1 partition. The preserve value is set to synthesis, which means packing, placement, and routing will be re-implemented.
Tcl Return:	The value of the previous preserve property.

process (run and manage project processes)

The *process* command runs and manages all processes within the current ISE project.

run (run process task)

The *process run* command runs the synthesis and implementation tools based on the specified process task. Process tasks are entered on the command line as strings distinguished by double quotes (“). The exact text representation of the task in Project Navigator is required. For a complete list of process tasks, see Table 3-4.

```
% process run <process_task> [-instance <instance_name>] [-force rerun|rerun_all]
```

process is the name of the Xilinx Tcl command.

run is the name of the process subcommand.

process_task specifies the name of one of the process tasks to run. Process tasks are listed in the Process window in Project Navigator. Note that the list of available processes changes, based on the source file you select. You can also use the *project get_processes* command to view a list of available processes. See the *process get_processes* command for more information. Process tasks vary by device family.

instance_name specifies the name of the instance to force re-implementation of. This is only needed for processes that do not use the entire design.

-force is the command to force the re-implementation of the specified process, regardless of the partition preserve setting. See the *partition set* command for more information on setting preservation levels.

rerun reruns the processes and updates input data as necessary, by running any dependency processes that are out-of-date.

rerun_all reruns the processes and all dependency processes back to the source data, as defined by the specified process goal. All processes are run whether they are out of date or not.

Example:	**`% process run “Implement Design” -force rerun_all`**
Description:	In this example, the *process run* command is used to force the re-implementation of the entire design, regardless of whether all source files are up-to-date or not.
Tcl Return:	True if the process was successful; false otherwise.

The following table lists the Tcl-supported process tasks, which are based on the GUI names in Project Navigator. Process tasks are entered as text strings, distinguished by double quotes (“), as shown in the table. Note that the source file determines what processes are available. This table lists all processes in order, beginning with synthesis.

Table 3-4: **Process Tasks**

“Synthesize - XST”
“Check Syntax”
“Generate Post-Synthesis Simulation Model”
“Implement Design”

Table 3-4: **Process Tasks**

“Translate”
“Map”
“Generate Post-Map Static Timing”
“Generate Post-Map Simulation Model”
“Place & Route”
“Generate Primetime Netlist”
“Generate Post-Place & Route Static Timing”
“Generate Post-Place & Route Simulation Model”
“Generate IBIS Model”
“Back-Annotate Pin Locations”
“Generate Programming File”

project (create and manage projects)

The *project* command creates and manages ISE projects. A project contains all files and data related to a design. You can create a project to manage all of the design files and to enable different processes to work on the design.

```
% project <subcommand>
```

clean (remove system-generated project files)

The *project clean* command removes all of the temporary and system-generated files in the current ISE project. It does not remove any source files, like Verilog or VHDL, nor does it remove any user-modified files. For example, system generated design and report files like the NCD (.ncd) and map report (.mpr) are removed with the *project clean* command, unless they have been user-modified.

```
% project clean
```

project is the name of the Xilinx Tcl command.

clean is the name of the project subcommand.

Example:	**`% project clean`**
Description:	In this example, the current ISE project is cleaned. All temporary and system generated files are removed, including design and report files, unless these have been user-modified.
Tcl Return:	True if the project is cleaned successfully; false otherwise.

Caution! The *project clean* command permanently deletes all system-generated files from the current ISE project. These files include the NGD and NCD files generated by the implementation tools.

close (close the ISE project)

The *project close* command closes the current ISE project. It is not necessary to specify the name of the project to close, since only one ISE project can be open at a time.

```
% project close
```

project is the name of the Xilinx Tcl command.

close is the name of the project subcommand.

Example:	**% project close**
Description:	In this example, the current ISE project is closed.
Tcl Return:	True if the project is closed successfully; false otherwise.

get (get project properties)

The *project get* command returns the value of the specified project-level property or batch application option.

```
% project get <option_name|property_name>
```

project is the name of the Xilinx Tcl command.

get is the name of the project subcommand.

option_name specifies the name of the batch application option you wish to get the value of. For example, Map Effort Level. Batch application options are entered as strings distinguished by double quotes (“). The exact text representation of the option in Project Navigator is required. For a complete list of project properties and options, see the “Project Properties and Options” section of this chapter.

property_name specifies the name of the property you wish to get the value of. Valid properties names are family, device, package, speed, and top.

Example:	**% project get speed**
Description:	In this example, the value of the speed grade that was set with the *project set speed* command is returned.
Tcl Return:	The property value as a text string. In this example, the device speed grade is returned.

get_processes (get project processes)

The *project get_processes* command lists the available processes for the specified instance.

```
% project get_processes [-instance <instance_name>]
```

project is the name of the Xilinx Tcl command.

get_processes is the name of the project subcommand.

-instance limits the properties listed to only those of the specified instance. If no instance is specified, the top-level instance is used by default.

instance_name specifies the name of the instance you wish to know the available processes for.

Example:	**% project get_processes [-instance Inst_dcm1]**
Description:	In this example, the *project get_processes* command is used to list all of the available processes for only the instance, Inst_dcm1.
Tcl Return:	The available processes as a text string. In this example, a list of processes for Inst_dcm1 is returned.

new (create a new ISE project)

The *project new* command creates a new ISE project.

```
% project new <project_name>
```

project is the name of the Xilinx Tcl command.

new is the name of the project subcommand.

project_name specifies the name for the ISE project you wish to create.

Example:	**% project new watchver.ise**
Description:	In this example, a new ISE project named watchver.ise is created in the current directory.
Tcl Return:	The name of the new ISE project.

open (open an ISE project)

The *project open* command opens an existing ISE project. If the project does not exist, an error message to create a new project with the *project new* command appears.

```
% project open <project_name>
```

project is the name of the Xilinx Tcl command.

open is the name of the project subcommand.

project_name specifies the name for the ISE project you wish to open.

Example:	**% project open watchver.ise**
Description:	In this example, the watchver.ise project in the current directory is opened.
Tcl Return:	The name of the open ISE project.

properties (list project properties)

The *project properties* command lists all of the project properties for the specified process or instance.

```
% project properties [-process <process_name>][-instance <instance_name]
```

project is the name of the Xilinx Tcl command.

properties is the name of the project subcommand.

-process <process_name> limits the properties listed to only those for the specified process. By default, the properties for all synthesis and implementation processes are listed. You can also specify *all* to list the properties for all project processes.

-instance <instance_name> limits the properties listed to only those of the specified instance. If no instance name is specified, the properties for the top-level instance are listed. You can also specify *top* to specify the top-level instance.

You can

Example:	**% project properties -process all**
Description:	In this example, the *project properties* command is used to list the properties for all of the available processes for the current ISE project.
Tcl Return:	The available process properties as a Tcl list. In this example, a list of all process properties.

Note: To get processes information for a specific instance, use the *project get_processes* command. To get property information for specific properties like family, device, and speed, see the project set options in this section for more information.

set (set project properties, values, and options)

The *project set* command is used to set properties and values for the current ISE project. In addition to setting family and device-specific properties and values, the *project set* command is also used to set options for the batch application tools, including XST, NGDBuild, MAP, PAR, TRACE, and BitGen.

The *set* subcommand uses two arguments. The first argument assigns the name of the property or variable; and the second argument assigns the value.

```
% project set <property_name> <property_value>
```

project is the name of the Xilinx Tcl command.

set is the name of the project subcommand.

property_name specifies the name of the property, variable or batch application option.

property_value specifies the value of the property, variable, or batch application option.

Note: Some batch application options only work when other options are specified. For example, in XST, the Synthesize Constraints File option only works if the Use Synthesis Constraints File option is also specified.

Example:	**% project set "Map Effort Level" high**
Description:	In this example, the *project set* command is used to set the map effort level to high. Map Effort Level is the name of the MAP option. High is the value set for the option.
Tcl Return:	The previous value of the newly set option. In this example, the Tcl return would be medium, if the option value was previously set to medium.

Note: Batch application options are entered as strings distinguished by double quotes ("). The exact text representation of the option (or property) in the Project Navigator GUI is required. For a complete list of project properties and options, see the "Project Properties and Options" section of this chapter.

set device (set device)

The *project set device* command specifies the target device for the current ISE project.

Note: A list of available devices can be viewed in the Project Properties dialog box in Project Navigator, or by utilizing the unique prefixes supported by Xilinx Tcl commands. For example, type **project set device V** to get an error message that enumerates all Virtex devices. Optionally, you can specify the *partgen –arch* command. From the Tcl prompt (%), type **partgen -h** for help using this command.

```
% project set device <device_name>
```

project is the name of the Xilinx Tcl command.

set device is the name of the project subcommand.

device_name specifies the target device for the current ISE project.

Example:	**% project set device xc2vp2**
Description:	In this example, the device for the current project is set to xc2vp2.
Tcl Return:	The previous value. In this example, the previous device setting is returned.

Note: You must first use the *set family* command to set the device family before using this command to set the device.

set family (set device family)

The *project set family* command specifies the device family for the current ISE project.

Note: A list of available devices can be viewed in the Project Properties dialog box in Project Navigator, or by utilizing the unique prefixes supported by Xilinx Tcl commands. For example, type **project set device V** to get an error message that enumerates all Virtex devices. Optionally, you can specify the *partgen –arch* command. From the Tcl prompt (%), type **partgen -h** for help using this command.

```
% project set family <device_family_name>
```

project is the name of the Xilinx Tcl command.

set family is the name of the project subcommand.

device_family_name specifies the device family to use with the current ISE project.

Example:	**% project set family Virtex2p**
Description:	In this example, the device family for the current project is set to Virtex2p.
Tcl Return:	The previous value. In this example, the previous device family setting is returned.

set package (set device package)

The *project set package* command specifies the device package for the current ISE project.

Note: A list of available devices can be viewed in the Project Properties dialog box in Project Navigator, or by utilizing the unique prefixes supported by Xilinx Tcl commands. For example, type **project set device V** to get an error message that enumerates all Virtex devices. Optionally, you can specify the *partgen –arch* command. From the Tcl prompt (%), type **partgen -h** for help using this command.

```
% project set package <package_name>
```

project is the name of the Xilinx Tcl command.

set package is the name of the project subcommand.

package_name specifies the target device package for the current ISE project.

Example:	**% project set package fg256**
Description:	In this example, the device package for the current project is set to fg256.
Tcl Return:	The previous value. In this example, the previous device package setting is returned.

set speed (set device speed)

The *project set speed* command specifies the device speed for the current ISE project.

Note: A list of available devices can be viewed in the Project Properties dialog box in Project Navigator, or by utilizing the unique prefixes supported by Xilinx Tcl commands. For example, type **project set device V** to get an error message that enumerates all Virtex devices. Optionally, you can specify the *partgen –arch* command. From the Tcl prompt (%), type **partgen -h** for help using this command.

```
% project set speed <speed_grade>
```

project is the name of the Xilinx Tcl command.

set speed is the name of the project subcommand.

speed_grade specifies the speed for the target device of the current ISE project.

Example:	**% project set speed -7**
Description:	In this example, the device speed for the current project is set to -7.
Tcl Return:	The previous value. In this example, the previous speed grade setting is returned.

set top (set the top-level module/entity)

The *project set top* command specifies the top-level module or entity in the design hierarchy. To use this command, you must first add the module or entity to your project with the *xfile add* command.

```
% project set top <module_name>
```

project is the name of the Xilinx Tcl command.

set top is the name of the project subcommand.

module_name specifies the name for the top-level module for Verilog and EDIF-based designs.

For VHDL designs, you must specify the architecture name and the entity name using the following syntax:

```
% project set top <architecture_name> [entity_name]
```

Example:	**% project set top pong_top**
Description:	In this Verilog example, the *project set top* command is used to set pong_top as the top-level module in the design hierarchy.
Tcl Return:	The name of the previous top-level module.

timing_analysis (generate timing analysis reports)

The *timing_analysis* command generates static timing analysis reports for an implemented design.

```
% timing_analysis <subcommand> <analysis_name>
```

delete (delete timing analysis)

The *timing_analysis delete* command removes a previously created analysis from the current ISE project.

```
% timing_analysis delete <analysis_name>
```

timing_analysis is the name of the Xilinx Tcl command.

delete is the name of the timing_analysis subcommand.

analysis_name specifies the name of the analysis to delete.

Example:	**% timing_analysis delete stopwatch_timing**
Description:	In this example, the *timing_analysis delete* command is used to remove the stopwatch_timing analysis, which was previously created with the *timing_analysis new* command.
Tcl Return:	0 if the analysis was deleted, 1 otherwise.

disable_constraints (disable timing constraints)

The *timing_analysis disable_constraints* command disables a physical timing constraint from the timing analysis.

```
timing_analysis disable_constraints <analysis_name> <timing_constraint_specs>
```

timing_analysis is the name of the Xilinx Tcl command.

disable_constraints is the name of the timing_analysis subcommand.

analysis_name specifies the name of an analysis created with the *timing_analysis new* command.

timing_constraint_specs specifies the names and specs to be disabled for analysis.

Example:	**% timing_analysis disable_constraints stopwatch_timing "TS_clk=PERIOD TIMINGROUP\"sclk\"20 ns HIGH 50.00000%;"**
Description:	In this example, the specified timing constraints were disabled for the analysis.
Tcl Return:	Number of constraints that were disabled.

disable_cpt (disable components for path tracing control)

The *timing_analysis disable_cpt* command disables components associated with a path tracing control for a timing path analysis.

```
% timing_analysis disable_cpt <analysis_name> <cpt_symbol> <component_name>
```

timing_analysis is the name of the Xilinx Tcl command.

disable_cpt is the name of the timing_analysis subcommand.

analysis_name specifies the name of the analysis generated previously with the *timing_analysis new* command.

cpt_symbol specifies a path tracing control that determines whether associated components are considered for the timing path analysis.

component_name specifies the name of the components to be disabled.

Example:	**% timing_analysis disable_cpt stopwatch_timing reg_sr_clk "ureg_1 ureg_2 ureg_3"**
Description:	In this example, the specified components, ureg_1, ureg_2, and ureg_3 are disabled for the timing path analysis.
Tcl Return:	Number of components that were disabled.

enable_constraints (enable constraints for analysis)

The *timing_analysis enable_constraints* command enables the specified timing constraints for the analysis.

```
% timing_analysis enable_constraints <analysis_name> <timing_constraint_specs>
```

timing_analysis is the name of the Xilinx Tcl command.

enable_constraints is the name of the timing_analysis subcommand.

analysis_name specifies the name of the analysis generated previously with the *timing_analysis new* command.

timing_constraint_specs specifies the names and specs of the constraints to be enabled for analysis.

Example:	**% timing_analysis enable_constraints stopwatch_timing "TS_clk=PERIOD TIMEGROUP\"sclk\" 20 ns HIGH 50.00000%;"**
Description:	In this example, the specified timing constraint is enabled for analysis.
Tcl Return:	Number of enabled constraints.

enable_cpt (enable components for path tracing control)

The *timing_analysis enable_cpt* command enables specified components associated with a path tracing control for a timing path analysis.

```
% timing_analysis enable_cpt <analysis_name> <cpt_symbol> <component_name>
```

timing_analysis is the name of the Xilinx Tcl command.

enable_cpt is the name of the timing_analysis subcommand.

analysis_name specifies the name of the analysis generated previously with the *timing_analysis new* command.

cpt_symbol specifies a path tracing control that determines whether associated components are considered for the timing path analysis.

component_name specifies the name of the components to enable.

Example:	**% timing_analysis enable_cpt stopwatch_timing reg_sr_clk "ureg_1 ureg_2 ureg_3"**
Description:	In this example, the specified components, ureg_1, ureg_2, and ureg_3 are enabled for the timing path analysis.
Tcl Return:	Number of components that were enabled.

get (get analysis property)

The *timing_analysis get* command returns the value of the specified property.

```
% timing_analysis get <analysis_name> <analysis_property>
```

timing_analysis is the name of the Xilinx Tcl command.

get is the name of the timing_analysis subcommand.

analysis_name specifies the name of the analysis generated previously with the *timing_analysis new* command.

analysis_property specifies the name of the analysis property to get the value of. See Table 3-5 for a list of analysis properties.

Example:	**% timing_analysis get stopwatch_timing analysis_speed**
Description:	In this example, *the timing_analysis get* command is used to return the speed grade that is currently set for the stopwatch_timing analysis.
Tcl Return:	The value of the specified property.

The following table lists the analysis properties for the *timing_analysis* command. A description of each analysis property is also included in the table.

Table 3-5: **Timing Analysis Properties and Descriptions**

Analysis Property	Description
analysis_type	Determines which of the following analysis types will be run: *auto_generated*—for an analysis that discards any constraints from the UCF/PCF and instead uses constraints automatically generated by the timing wizard. *clock_io*—for an analysis that discards any constraints from the UCF/PCF and uses custom constraints as generated by the set_constraints subcommand. *endpoints*—for an analysis that discards any constraints from the UCF/PCF and uses custom constraints as generated by the set_endpoints subcommand. *timing_constraint*—for an analysis that uses the constraints form the UCF/PCF. *net*—for an analysis on nets as specified by the set_query subcommand. *timegroup*—for an analysis on timegroups as specified by the set_query subcommand.
omit_user_constraints	Specifies whether to omit all timing constraints from the UCF/PCF.
analyze_unconstrained_paths	Specifies whether paths not covered by any constraint should be analyzed and shown in the timing report.
analysis_temperature	Specifies the prorating temperature for the analysis.
analysis_voltage	Specifies the prorating voltage for the analysis.
analysis_speed	Specifies the speed grade for the analysis.
report_name	Specifies the name for the report (XML or ASCII).
report_format	Specifies the format for the report.
report_datasheet	Specifies whether the datasheet section is generated for the timing report.
report_timegroups	Specifies whether the timegroups table is generated for the timing report.
paths_per_constraint	Specifies how many paths per constraint are reported.

Table 3-5: **Timing Analysis Properties and Descriptions**

Analysis Property	Description
show_longest_paths	CPLD report option that determines whether the longest paths are shown.
show_delay_over	CPLD report option that specifies only paths above the specified delay are shown in the report.
show_delay_under	CPLD report option that specifies that only paths below the specified delay are shown in the report.
display_info	Specifies whether Timing Analyzer is run in verbose mode.
display_physical_name	Specifies whether physical names of path elements in the timing report should be displayed in the Timing Analyzer report view.
display_site_location	Specifies whether site locations of path elements in the timing report should be displayed in the Timing Analyzer report view.
display_statistics	Specifies whether the statistic section of the timing report is shown in the Timing Analyzer report view.

new (new timing analysis)

The *timing_analysis new* command sets up a new analysis or query on an implemented design in the current ISE project. The *timing_analysis set* command is used to set properties and values for the new analysis. See "set (set analysis properties)" for more information.

```
% timing_analysis new analysis|query [-name <analysis_name>]
```

timing_analysis is the name of the Xilinx Tcl command.

new is the name of the timing_analysis subcommand.

analysis, if specified, sets up a timing analysis.

query, if specified, sets up a net or timegroup analysis.

-name <analysis_name> specifies the name for the analysis. If the *-name* command is used, but no name is specified, an analysis is generated and has the name of the top-level module.

Example:	**% timing_analysis new analysis [-name stopwatch_timing]**
Description:	In this example, the *timing_analysis new* command is used to create a timing analysis named stopwatch_timing.
Tcl Return:	A new timing analysis.

reset (reset path filters and constraints)

The *timing_analysis reset* command resets all existing path filters and custom constraints for the analysis.

```
% timing_analysis reset <analysis_name>
```

timing_analysis is the name of the Xilinx Tcl command.

reset is the name of the timing_analysis subcommand.

analysis_name specifies the name of the analysis previously created with the *timing_analysis new* command.

Example:	**% timing_analysis reset stopwatch_timing**
Description:	In this example, the *timing_analysis reset* command is used to reset all of the path filters and any custom constraints in the stopwatch_timing analysis.
Tcl Return:	True if all path filters were cleared successfully, false otherwise.

run (run analysis)

The *timing_analysis run* command executes an analysis and returns the name of the timing, net, or timegroup report file. The analysis is based on the property settings assigned with the *timing_analysis set* command. An analysis is first created with the *timing_analysis new* command. See "set (set analysis properties)" and "new (new timing analysis)" for more information.

```
% timing_analysis run <analysis_name>
```

timing_analysis is the name of the Xilinx Tcl command.

run is the name of the timing_analysis subcommand.

analysis_name specifies the name of the analysis previously created with the *timing_analysis new* command.

Example:	**% timing_analysis run stopwatch_timing**
Description:	In this example, the *timing_analysis run* command is used to execute a timing analysis and create a new analysis report.
Tcl Return:	Name of the timing analysis.

saveas (save analysis report)

The *timing_analysis saveas* command saves the analysis to a specified file.

```
% timing_analysis saveas <analysis_name> <new_file_name>
```

timing_analysis is the name of the Xilinx Tcl command.

saveas is the name of the timing_analysis subcommand.

analysis_name specifies the name of the analysis previously created with the *timing_analysis new* command.

new_file_name specifies the file name for the analysis report.

Example:	`% timing_analysis saveas stopwatch_timing stopwatch_report`
Description:	In this example, the *timing_analysis saveas* command is used to save the stopwatch_timing analysis to a report file named stopwatch_report.
Tcl Return:	Name of the report file.

set (set analysis properties)

The *timing_analysis set* command is used to set properties and values for an analysis.

```
% timing_analysis set <analysis_name> <property> <value>
```

timing_analysis is the name of the Xilinx Tcl command.

set is the name of the timing_analysis subcommand.

analysis_name specifies the name of the analysis previously created with the *timing_analysis new* command.

property specifies the analysis property that you wish to set the value for. See Table 3-5 for a list of analysis properties.

value specifies the value for the specified property.

Example:	**`% timing_analysis set stopwatch_timing analysis_speed -11`**
Description:	In this example, the *timing_analysis set* command is used to set the value of the analysis_speed property to -11, for the stopwatch_timing analysis.
Tcl Return:	Previous value of the specified property.

set_constraint (set constraint for custom analysis)

The *timing_analysis set_constraint* command is used to set constraints for a custom analysis.

```
% timing_analysis set_constraint <analysis_name> <constraint_type>
<constraint_details>
```

timing_analysis is the name of the Xilinx Tcl command.

set_constraint is the name of the timing_analysis subcommand.

analysis_name specifies the name of the analysis previously created with the *timing_analysis new* command.

constraint_type specifies the type of constraint to set for the custom analysis. There are four constraint types:

- maxdelay—pad-to-pad maximum delay
- period—period constraint on clock pad

- padgroup—group definition on pad set
- offset—offset in/out constraint

constraint_details specifies the details for the specified constraint type as follows:

- maxdelay—timing in ns
- period—time in ns and clock pad
- padgroup—group name and pads to be included in group
- offset—[timegroup] P2S|C2P <*clock_pad_name*> <*time_in_ns*> [C2P|P2S <*clock_pad_name*> <*time_in_ns*>] [<"data pad or pad group">]

  Note: There are two types of offset constraints: P2S, which is pad-to-synchronous (offset in constraint), and C2P, which is clock-to-pad (offset out constraint).

Example:	**% timing_analysis set_constraint stopwatch_timing period "13 sclk"**
Description:	In this example, a period constraint is set for the stopwatch_timing analysis. Note that the constraint details, "13 sclk", are entered as a text string.
Tcl Return:	1 if the constraint was set successfully, 0 otherwise.

set_endpoints (set source and destination endpoints)

The *timing_analysis set_endpoints* command sets source and destination endpoints for a path endpoints analysis.

```
% timing_analysis set_endpoints <analysis name> <qualifier> <category> <items>
```

timing_analysis is the name of the Xilinx Tcl command.

set_endpoints is the name of the timing_analysis subcommand.

analysis_name specifies the name of the analysis previously created with the *timing_analysis new* command.

qualifier specifies whether source (from) or destination (to) points are being set for custom analysis.

category specifies the resources type for the sources and destinations. These are family dependent.

items specifies the resource items to be put into path endpoint sources or destinations.

Example:	**% timing_analysis set_endpoints stopwatch_timing from ffs ***
Description:	In this example, the asterisk (*) was used as a wildcard to add all items in the flip-flop category to the sources for the path endpoints analysis.
Tcl Return:	1 if the endpoints were set successfully, 0 otherwise.

set_filter (set filter for analysis)

The *timing_analysis set_filter* command sets a net filter to exclude or include the specified nets from a path analysis.

```
% timing_analysis set_filter <analysis_name> <filter_type> <filter_value>
<filter_items>
```

timing_analysis is the name of the Xilinx Tcl command.

set_filter is the name of the timing_analysis subcommand.

analysis_name specifies the name of the analysis previously created with the *timing_analysis new* command.

filter_type specifies the type of analysis filter. Supported filter types are net and timegroup.

filter_value specifies the value for the filter type. Net filter values are include and exclude.

filter_items specifies the items to be filtered. For the net filter type, these are the names of nets in the current design.

Example:	**% timing_analysis set_filter stopwatch_timing nets exclude "ureg_net_1 ureg_net_2 ureg_net_3"**
Description:	In this example, the specified nets are excluded from the stopwatch_timing analysis. Note that the nets to exclude are entered as text strings, which are distinguished by double quotes (“).
Tcl Return:	1 if the filter is set successfully; 0 otherwise.

set_query (set up net or timegroup report)

The *timing_analysis set_query* command sets up a report that shows net delays and fanouts, or blocks associated with timegroups.

```
% timing_analysis set_query <analysis_name> <query_type> <query_items>
```

timing_analysis is the name of the Xilinx Tcl command.

set_query is the name of the timing_analysis subcommand.

analysis_name specifies the name of the analysis previously created with the *timing_analysis new* command.

query_type specifies the type of query. Supported queries are:

net—generates a report that shows the delay details for the specified query items.

timegroup—generates a report that shows blocks of the specified timegroups.

query_items specifies the items to query. For example, -ld <*number*> specifies how many longest delay nets should be displayed in the report; -hf <*number*> specifies how many highest fanout nets should be displayed in the report. See the example below.

Example:	**% timing_anlaysis run stopwatch_timing** **% timing_analysis set_query stopwatch_timing net "clk_net1 clk_net2" -ld 2 -hf 5**
Description:	In this example, a query is set up to report 2 nets with the longest delay and 5 nets with the highest fanout. Delay details on the query items (clk_net1 and clknet_2) are reported in the detailed nets section of the report. ***Note:*** The net report is only generated after the *timing_analysis run* command is used to set up the query.
Tcl Return:	1 if the command was executed successfully; 0 otherwise.

show_settings (generate settings report)

The *timing_analysis show_settings* command generates a settings report based on the analysis settings.

```
% timing_analysis show_settings <analysis_name>
```

timing_analysis is the name of the Xilinx Tcl command.

show_settings is the name of the timing_analysis subcommand.

analysis_name specifies the name of the analysis previously created with the *timing_analysis new* command.

Example:	**% timing_analysis show_settings stopwatch_timing**
Description:	In this example, a settings report is generated for the stopwatch_timing analysis.
Tcl Return:	The name of the settings report.

xfile (manage project files)

The *xfile* command is used to manage all of the source files within an ISE project. Use the *xfile* command to add, remove, and get information on any source files in the current ISE project.

```
% xfile <subcommand> <file_name>
```

add (add file to project)

The *xfile add* command specifies the name of the file to add to the current ISE project. Files can be added to a project in any order.

```
% xfile add <file_name>
```

xfile is the name of the Xilinx Tcl command.

add is the name of the xfile subcommand.

file_name specifies the name of the source file(s) you wish to add to the current ISE project. Path names and wildcards can be used to specify one or more files to add to the project. Tcl commands support two wildcard characters: asterisk (*) to indicate multiple characters, or a question mark (?) to indicate a single character.

Example:	**% xfile add *.vhd /mysource/mysub_dir timing.ucf**
Description:	In this example, the *xfile add* command is used to add all of the VHDL source files and the timing.ucf file to the current ISE project.
Tcl Return:	The name of the added file(s).

get (get project file properties)

The *xfile get* command returns information on the specified project file and its properties. There are two properties supported for this command: name and timestamp. For example, if *name* is the specified property, the Tcl return is the full name of the specified file. If *timestamp* is the specified property, the Tcl return is the timestamp of when the file was first added to the project with the *xfile add* command.

```
% xfile get <file_name> <name|timestamp>
```

xfile is the name of the Xilinx Tcl command.

get is the name of the xfile subcommand.

file_name specifies the name of the source file to get the name or timestamp information on.

name if specified, returns the full path of the current project and the name of the specified file.

timestamp if specified, returns the timestamp of when the file was first added to the project with the *xfile add* command.

Example:	**% xfile get timestamp stopwatch.vhd**
Description:	In this example, the *xfile get* command is used to get the timestamp information for the stopwatch.vhd file.
Tcl Return:	The value of the specified property as a text string. In this example, the timestamp information of when the file was added to the project.

remove (remove file from project)

The *xfile remove* command removes the specified file from the current ISE project.

```
% xfile remove <file_name>
```

xfile is the name of the Xilinx Tcl command.

remove is the name of the xfile subcommand.

file_name specifies the name of the file you wish to remove from the project.

Example:	**% xfile remove stopwatch.vhd**
Description:	In this example, the stopwatch.vhd file is removed from the current ISE project.
Tcl Return:	True if the file was removed; false otherwise.

Note: When you remove a file, objects within the current ISE project may be invalidated (e.g., partitions and instances).

Tcl Commands for Advanced Scripting

Xilinx Tcl commands for advanced scripting use objects and collections. An object can be any element in an ISE project, like an instance, file, or process. Collections return groups of objects, based on values that you assign to object and collection variables.

In Tcl, the *set* command is used to assign a value to a variable, which is returned with the dollar sign ($) syntax, as shown in many examples throughout this section. It is not necessary to declare a Tcl variable before it is used. When the variable does not exist, it is created when the command is executed.

The *collection* command and its relative subcommands are used to create and manage large groups of objects. The *search* command is used with the *collection* command to define the value of the collection.

This section describes the Xilinx Tcl commands for advanced scripting. To view a sample script of how these commands are used, see the "Sample Tcl Script for Advanced Scripting" at the end of this chapter.

collection (create and manage a collection)

A collection is a group of Xilinx Tcl objects, similar to a list, that is exported to the Tcl interface. The *collection* command, in conjunction with its subcommands, is used to create and manage the objects in a specified collection.

A collection is referenced in Tcl by a collection variable, which is defined with the *collection set* command. Technically, the value of the collection variable is the collection.

The following syntax shows the *collection* command and its subcommands. Please refer to the description of each *collection* subcommand for an example of how the subcommand is used. Command line syntax is unique to each subcommand.

```
% collection <subcommand> <optional_arguments>
```

append_to (add objects to a collection)

The *collection append_to* command adds objects to a collection. This command treats a specified collection variable as a collection and appends all of the objects returned from a search, or from another collection, to the collection. If the collection variable does not exist, then it is created when the command is executed.

```
% collection append_to <collection_variable> <objects_to_append> [-unique]
```

collection is the name of the Xilinx Tcl command.

append_to is the name of the collection subcommand.

collection_variable specifies the name of the collection variable, which references the collection. If the collection variable does not exist, then it is created.

objects_to_append specifies an object or a collection of objects to be added to the collection.

-unique optionally adds only objects that are not already in the collection. If the -unique option is not used, then duplicate objects may be added to the collection.

Example:	`% collection append_to colVar [search * -type instance]`
Description:	In this example, the *collection append_to* command is used to create a new collection variable named colVar. The nested *search* command returns a collection of all the instances in the current design. These instances are objects that are added to the collection, referenced by the colVar collection variable.
Tcl Return:	A collection of objects.

copy (copy a collection)

The *collection copy* command creates a duplicate of an existing collection. Alternatively, you can have more than one collection variable referencing a collection, rather than copying it. See Example 1 below.

In most cases, a second reference to a collection is all that is needed. However, if a separate copy is required, use the *collection copy* command to create the new collection as shown in Example 2 below.

```
collection copy <collection_variable>
```

collection is the name of the Xilinx Tcl command.

copy is the name of the collection subcommand.

collection_variable specifies the name of the collection to copy.

Example 1:	`% set colVar_1 [search * -type instance]` `% set colVar_2 $colVar_1`
Description:	In this example, the Tcl *set* command in the first line creates a collection assigned to the collection variable colVar_1. The second line creates a second collection variable, colVar_2, that references the value of colVar_1, which is the first collection. There is still only one underlying collection referenced. Any changes made to colVar_1 will be visible in colVar_2, and if colVar1_1 is changed, then colVar_2 continues to reference the same underlying collection.
Tcl Return:	A new variable reference to the collection.

Example 2:	**% set colVar_2 [collection copy $colVar_1]**
Description:	In this example, the *set* command creates the collection variable colVar_2. The nested *collection copy* command makes a duplicate of the colVar_1 collection and assigns it to the colVar2 collection variable, making it a completely separate collection.
Tcl Return:	A new collection.

equal (compare two collections)

The *collection equal* command compares the contents of two collections. Collections are considered equal when the objects in both collections are the same. If the same objects are in both collections, the result is 1. If the objects in the compared collections are different, then the result is 0. By default, the order of the objects does not matter. Optionally, the *order_dependent* command can be specified for the order of the objects to be considered.

```
% collection equal <colVar_1> <colVar_2> [-order_dependent]
```

collection is the name of the Xilinx Tcl command.

equal is the name of the collection subcommand.

colVar_1 specifies the base collection for the comparison.

colVar_2 specifies the collection to compare with the base collection.

order_dependent optionally specifies that the collections are considered different when the order of the objects in both collections are not the same.

Note: When two empty collections are compared, they are considered identical and the result is 1.

Example:	**% set colVar_1 [search * -type instance]** **% set colVar_2 [search /top/T* -type instance]** **% collection equal $colVar_1 $colVar_2**
Description:	In this example, the contents of two collections are compared. First, the Tcl *set* command is used to assign a collection of instances to the collection variable colVar_1; then another collection of filtered instance names is assigned to the collection variable colVar_2. The *collection equal* command is used to specify the two collections to compare. The dollar sign ($) syntax is used to obtain the values of the collection variables. In this case, the values of colVar_1 and colVar_2 to determine if they are equal.
Tcl Return:	0 if the collections are not the same, 1 if the collections are the same.

foreach (iterate over elements in a collection)

The *collection foreach* command iterates over each object in a collection through an iterator variable. The iterator variable specifies the collection to interate over and the set of commands or script to apply at each iteration.

```
% collection foreach <iterator_variable> <collection_variable> {body}
```

collection is the name of the Xilinx Tcl command.

foreach is the name of the collection subcommand.

iterator_variable specifies the name of the iterator variable.

collection_variable specifies the name of the collection to iterate through.

body specifies a set of commands or script to execute at each iteration.

Example:	**% set colVar [search * -type instance]** **% collection foreach itr $colVar {** **puts [object name $itr]}**
Description:	In this example, the *set* command is used in the first line to assign a collection of instances to the colVar collection variable. In the second line, the *collection foreach* command is used to iterate over each object in the colVar collection. *itr* is the name of the iterator variable. Curly braces *{ }* enclose the body, which is the script that executes at each iteration. Note that the *object name* command is nested in the body to return the value of the iterator variable, which is an instance in this case.
Tcl Return:	An integer return of the number of times the script was executed.

Caution! You cannot use the standard Tcl-supplied *foreach* command to iterate over collections. You must use the Xilinx-specific collection *foreach* command. Using the Tcl-supplied *foreach* command may cause the collection to be deleted.

get (get collection property)

The *collection get* command returns the value of the specified collection property. Collection properties and values are assigned with the *collection set* command.

```
collection get <property_name>
```

collection is the name of the Xilinx Tcl command.

get is the name of the collection subcommand.

property_name specifies the name of the property you wish to get the value of. Valid property names for the *collection get* command are display_line_limit and display_type.

Example:	**% collection get display_type**
Description:	In this example, the *collection get* command is used to get the current setting of the display_type property.
Tcl Return:	The set value of the specified property.

Note: See also the *collection set* command for more information on property values for the *collection* command.

index (extract a collection object)

Given a collection and an index into it, the *collection index* command extracts the object at the specified index and returns the object, if the index is in range. The base collection is unchanged.

The range for an index is zero (0) to one less (-1) the size of the collection obtained with the *collection sizeof* command. See "sizeof (show the number of objects in a collection)."

```
collection index <collection_variable> <index>
```

collection is the name of the Xilinx Tcl command.

index is the name of the collection subcommand.

collection_variable specifies the name of the collection for index.

index specifies the index into the collection. Index values are 0 to -1, unless the size of the collection was defined with the *collection sizeof* command.

Example:	**% set colVar [search * -type instance]** **% set item [collection index $colVar 2]** **% object name $item**
Description:	In this example, the *collection index* command extracts the third object in the collection of instances. In the first line, the *set* command is used to create a collection variable named colVar. The nested *search* command defines the value of the collection for colVar, which in this case is all of the instances in the current design. In the second line, the *set* command is used to create a variable named item. The nested *collection index* command obtains the third object (starting with index 0, 1, 2 . . .) in the given collection. The last line of this example uses the *object name* command to return the value of the item variable, which is the specified value of index.
Tcl Return:	The object at the specified index.

Note: Xilinx-specific Tcl commands that create a collection of objects do not impose a specific order on the collection, but they do generate the objects in the same, predictable order each time. Applications that support sorting collections, can impose a specific order on a collection.

properties (list available collection properties)

The *collection properties* command displays a list of the supported properties for all collections in the current ISE project. You can set the value of any property with the *collection set* command.

```
% collection properties
```

collection is the name of the Xilinx Tcl command.

properties is the name of the collection subcommand. There are two collection properties: display_line_limit and display_type. These properties are supported with the *collection get* and *collection set* commands.

Example:	**% collection properties**
Description:	In this example, the *collection properties* command is used to display a list of available collection properties.
Tcl Return:	A list of available collection properties. In this example, display_line_limit and display_type.

Note: See the *collection get* command for a list of available properties.

remove_from (remove objects from a collection)

The *collection remove_from* command removes objects from a specified collection, modifying the collection in place. If you do not wish to modify the existing collection, first use the *collection copy* command to create a duplicate of the collection.

```
% collection remove_from <collection_variable> <objects_to_remove>
```

collection is the name of the Xilinx TCL command.

remove_from is the name of the collection subcommand.

collection_variable specifies the name of the collection variable.

objects_to_remove specifies a collection of objects, or the name of an object that you wish to remove from the collection.

Example:	**% set colVar_1 [search * -type instance]** **% set colVar_2 [search /stopwatch/s* -type instance]** **% set colVar_3 [collection remove_from colVar_1 $colVar_2]**
Description:	In this example, the *set* command is first used to create the collection variables colVar_1 and colVar_2. Assume that the values of these two variables are different. The last line of this example, creates a third collection variable, colVar_3 that contains all of the instances in colVar_1, but no instances in colVar_2.
Tcl Return:	The original collection modified by removed elements. In this example, the objects in colVar_2 are removed from colVar_1.

set (set the property for all collections)

The collection *set* command sets the specified property for all collection variables in the current ISE project.

```
% collection set <property_name> <property_value>
```

collection is the name of the Xilinx Tcl command.

set is the name of the collection subcommand.

property_name and property_value specify the property name and value for all of the collection variables in the current ISE project. There are two available property settings for the *collection set* command, they are:

display_line_limit–specifies the number of lines that can be displayed by a collection variable. This property setting is useful for very large collections, which may have thousands, if not millions of objects. The default value for this property is 100. The minimum value is 0. There is no maximum value limit for this property.

display_type–instructs Tcl to include the object type in the display of objects from any specified collection variable. Values for this property are true and false. By default, this option is set to false, which means object types are not displayed. See the example below.

Example:	**% collection set display_type true**
Description:	In this example, the *collection set* command is used to set the property name and value for all collection variables in the project. *display_type* is the name of the property setting. *true* specifies the value for the property.
Tcl Return:	The previous value of the property.

sizeof (show the number of objects in a collection)

The *collection sizeof* command returns the number of objects in the specified collection.

```
% collection sizeof <collection_variable>
```

collection is the name of the Xilinx Tcl command.

sizeof is the name of the collection subcommand.

collection variable specifies the name of the collection for Tcl to return the size of.

Example:	**% collection sizeof $colVar**
Description:	In this example, the *collection sizeof* command is used to return the size of the collection, which is referenced by the colVar collection variable.
Tcl Return:	An integer return of the number of items in the specified collection.

object (get object information)

The *object* command returns the name, type, or property information of any Xilinx Tcl object in the current ISE project. You can specify a single object or an object from a collection of objects.

```
% object <subcommand>
```

get (get object properties)

The *object get* command returns the value of the specified property.

```
% object get <object_name> <property_name>
```

object is the name of the Xilinx Tcl command.

name is the name of the object subcommand.

object_name specifies the object to get the property of. The object name will always be a Tcl variable. The *set* command is used to create a Tcl variable, as shown in the following example.

property_name specifies the name of one of the properties of an object. The properties of an object vary depending on the type of object. Use the *object properties* command to get a list of valid properties for a specific object.

Example:	**% set colVar [search * -type instance]** **% collection foreach obj $colVar {set objProps [object properties $obj] foreach prop $objProps {puts [object get $obj $prop]}}**
Description:	This example first runs a search to create a collection of all instances in the project. The second statement iterates through the objects in the collection. For each object, the list of available properties on the object are obtained by the *object properties* command. Then, the value of each property for each of the objects is returned.
Tcl Return:	The value of the specified property.

name (name of the object)

The *object name* command returns the name of any Xilinx object. This command is useful when manipulating a collection of objects, which may have been returned from a search. The *object name* command can be used to narrow the collection.

```
% object name <object_name>
```

object is the name of the Xilinx Tcl command.

name is the name of the object subcommand.

object_name specifies the object to return the name of. The object name will always be a string. The *set* command can be used to create a Tcl variable, as shown in the following example.

Example:	**% set colVar [search * -type instance]** **% object name [collection index $colVar 1]**
Description:	In this example, the *set* command is first used to create the colVar collection variable. The nested *search* command defines the value of the collection variable to be all instances in the current ISE project. In the second line, the *object name* command is used to get the name of the second object in the collection. The *collection index* command defines which object to get, *where*: *$colvar* specifies the collection to get the object from. *1* specifies the index into the collection. In this case, the second object in the collection is returned because index values start at 0. See the *collection index* command for more information.
Tcl Return:	The name of the object as a text string.

properties (list object properties)

The *object properties* command lists the available properties for the specified object.

```
% object properties <object_name>
```

object is the name of the Xilinx Tcl command.

name is the name of the object subcommand.

object_name specifies the object to list the properties of. The object name will always be a Tcl variable created with the Tcl set command.

Example:	**% set colVar [search * -type partition]** **% collection foreach obj $colVar {set objProps [object properties $obj] foreach prop $objProps {puts [object get $obj $prop]}}**
Description:	This example first runs a search to create a collection of objects. The second statement iterates through the objects in the collection. For each object, a list of available properties for the object are obtained with the object properties command. Then, the value of each property is returned for each object.
Tcl Return:	The available object properties as a text string.

type (type of object)

The object type command returns the type of any Xilinx object.

```
% object type <object_name>
```

object is the name of the Xilinx Tcl command.

name is the name of the object subcommand.

object_name specifies the object to return the type of. The object name will always be a Tcl variable. The set command is used to create a Tcl variable, as shown in the following example.

Example:	**% set colVar [search * -type instance]** **% object type [collection index $colVar 1]**
Description:	In this example, the *set* command is first used to create the colVar collection variable. The nested *search* command defines the value of the collection variable. In the second line, the *object type* command is used to get the type information for the object in the collection. The *collection index* command defines which object to get, where: *$colvar* specifies the collection to get the object from. *1* specifies the index into the collection. In this case, the second object in the collection is returned because index values start at 0. See the *collection index* command for more information.
Tcl Return:	The type of the object as a text string.

search (search and return matching objects)

The *search* command is used to search for specific design objects that match a specified pattern.

```
search <pattern_string> [-exactmatch][-matchcase][-regexp][-type <object_type>]
```

search is the name of the Xilinx Tcl command.

pattern_string specifies a string to be used for the search. In Tcl, a string is distinguished on the command line with double quotes (“). This string may contain regular expressions.

-exactmatch specifies that any matches found by the search, should match the pattern string exactly.

-matchcase specifies that the search is case-sensitive.

-regexp specifies that the pattern string uses regular expressions. If not specified, the matching is simple matching.

-type <object_type> specifies what type of design objects to search for. Supported search types are partition and instance.

Example:	`% search "/stopwatch" -type instance`
Description:	In this example, the *search* command is used to find all instances in the design.
Tcl Return:	A collection of objects that match the search criteria. If no matches are found, an empty collection is returned.

Note: Use the *collection foreach* command to list or get access to each object in a collection. See foreach (iterate over elements in a collection) for more information.

Project Properties and Options

This following tables list all of the available project properties and batch tool options. The first table lists all of the project properties. The remaining tables list all of the supported batch tool options, by Xilinx synthesis or implementation tool.

Batch tool options are listed as text strings, which are distinguished by double quotes (“) on the command line. The exact string representation, as shown in Project Navigator, is required.

Table 3-6: **Project Properties**

Property Name	Description
device	The device to use for the project.
family	The device family to use for the project.
generated_simulation_language	The language of the generated simulation netlist.
name	Name of the project.
package	The device package to use for the project.
speed	The device speed grade to use for the project.
synthesis_tool	The synthesis tool to use for this project. The default tool to use is XST.
top	The top-level design module or entity.
top_level_module_type	The module type of the top-level source.

Table 3-7: **XST Options**

Option Name	Synthesis Tool
“Add I/O Buffers”	XST
“Bus Delimiter”	XST
“Case Implementation Style”	XST
“Case”	CST
“Constrain Placement”	XST

Table 3-7: **XST Options**

Option Name	Synthesis Tool
“Cores Search Directories”	XST
“Cross Clock Analysis”	XST
“Custom Compile File List”	XST
“Decoder Extraction”	XST
“Equivalent Register Removal”	XST
“FSM Encoding Algorithm”	XST
“FSM Style”	XST
“Generate RTL Schematic”	XST
“Global Optimization Goal”	XST
“HDL INI File”	XST
“Hierarchy Separator”	XST
“Keep Hierarchy”	XST
“Library Search Order”	XST
“Logical Shifter Extraction”	XST
“Max Fanout”	XST
“Move First Flip-Flop Stage”	XST
“Move Last Flip-Flop Stage”	XST
“Mux Extraction”	XST
“Mux Style”	XST
“Number of Global Clock Buffers”	XST
“Number of Regional Clock Buffers”	XST
“Optimization Effort”	XST
“Optimization Goal”	XST
“Optimize Instantiated Primitives”	XST
“Other XST Command Line Options”	XST
“Pack I/O Registers into IOBs”	XST
“Priority Encoder Extraction”	XST
“RAM Extraction”	XST
“RAM Style”	XST
“Read Cores”	XST
“Register Balancing”	XST
“Register Duplication”	XST

Table 3-7: **XST Options**

Option Name	Synthesis Tool
“Resource Sharing”	XST
“ROM Extraction”	XST
“ROM Style”	XST
“Safe Implementation”	XST
“Shift Register Extraction”	XST
“Slice Packing”	XST
“Slice Utilization Ration”	XST
“Synthesis Constraints File”	XST
“Use Clock Enable”	XST
“Use DSP48”	XST
“Use Synchronous Reset”	XST
“Use Synchronous Set”	XST
“Use Synthesis Constraints File”	XST
“Verilog 2001”	XST
“Verilog Include Directories”	XST
“Work Directory”	XST
“Write Timing Constraints”	XST
“XOR Collapsing”	XST

Table 3-8: **NGDBuild Options**

Option Name	Implementation Tool
“Allow Unexpanded Blocks”	NGDBuild
“Allow Unmatched LOC Constraints”	NGDBuild
“Create I/O Pads from Ports”	NGDBuild
“Macro Search Path”	NGDBuild
“Netlist Translation Type”	NGDBuild
“Other NGDBuild Command Line Options”	NGDBuild
“Preserve Hierarchy on SubModule”	NGDBuild
“Use LOC Constraints”	NGDBuild
“Use Rules File for Netlister Launcher”	NGDBuild

Table 3-9: **MAP Options**

Option Name	Implementation Tool
“Allow Logic Optimization Across Hierarchy”	MAP
“CLB Pack Factor Percentage”	MAP
“Disable Register Ordering”	MAP
“Equivalent Register Removal”	MAP
“Extra Effort”	MAP
“Generate Detailed MAP Report”	MAP
“Global Optimization”	MAP
“Map Effort Level”	MAP
“Map Guide Design File (.ncd)”	MAP
“Map Guide Mode”	MAP
“Map Slice Logic into Unused Block RAMs”	MAP
“Map to Input Functions”	MAP
“Optimization Strategy (Cover Mode)”	MAP
“Other Map Command Line Options”	MAP
“Pack I/O Registers/Latches into IOBs”	MAP
“Perform Timing-Driven Packing and Placement”	MAP
“Register Duplication”	MAP
“Replicate Logic to Allow Logic Level Reduction”	MAP
“Retiming”	MAP
“Starting Placer Cost Table (1-100)”	MAP
“Trim Unconnected Signals”	MAP
“Use RLOC Constraints”	MAP

Table 3-10: **PAR Options**

Option Name	Implementation Tool
“Extra Effort (Highest PAR level only)”	PAR
“Generate Asynchronous Delay Report”	PAR
“Generate Clock Region Report”	PAR
“Generate Post-Place & Route Simulation Model”	PAR
“Generate Post-Place & Route Static Timing Report”	PAR
“Nodelist File (Unix Only)”	PAR
“Number of PAR Iterations (1-100)”	PAR
“Number of Results to Save (0-100)”	PAR
“Other Place & Route Command Line Options”	PAR
“PAR Guide Design File (.ncd)”	PAR
“PAR Guide Mode”	PAR
“Place & Route Effort Level (Overall)”	PAR
“Place and Route Mode”	PAR
“Place Effort Level (Overrides Overall Level)”	PAR
“Power Reduction”	PAR
“Router Effort Level (Overrides Overall Level)”	PAR
“Save Results in Directory (.dir will be appended)”	PAR
“Starting Placer Cost Table (1-100)”	PAR
“Use Bonded I/Os”	PAR
“Using Timing Constraints”	PAR

Example Tcl Scripts

This section provides two example Tcl scripts. The first example is a "Sample Tcl Script for General Usage". The second example is a "Sample Tcl Script for Advanced Scripting".

You can run these example Tcl scripts the following ways:

- Enter each statement interactively at the xtclsh prompt (%).

  You can access the xtclsh prompt (%) from the command line by typing **`xtclsh`**, or from the Tcl Console tab in Project Navigator.

- Save the statements in a script file with a .tcl extension. To run the Tcl script, type the following from the xtclsh prompt (%):

  ```
  % source <script_name>.tcl
  ```

Sample Tcl Script for General Usage

```
# open the project and set project-level properties
project new watchvhd.ise
project set family Virtex2P
project set device xc2vp2
project set package fg256
project set speed -7

# add all the source HDLs and ucf
xfile add stopwatch.vhd statmatch.vhd cnt60.vhd dcm1.vhd decode.vhd
xfile add tenths.vhd hex2led
xfile add watchvhd.ucf

# define all partitions
partition new /stopwatch/MACHINE
partition new /stopwatch/Inst_dcm1
partition new /stopwatch/XCOUNTER
partition new /stopwatch/decoder
partition new /stopwatch/sixty
partition new /stopwatch/lsbled
partition new /stopwatch/msbled

# get partition properties
set props [partition properties]
puts "Partition Properties :  $props"

# get top
set top [project get top]
```

```
# get project properties available
set props2 [project properties]
puts "Project Properties for top-level module $top" $props2

# inspect the current value for the following batch application options
set map_guide_mode [project get "MAP Guide Mode"]
puts "MAP Guide Mode = $map_guide_mode"
set par_guide_mode [project get "PAR Guide Mode"]
puts "PAR Guide Mode = $par_guide_mode"

# set batch application options :
# 1. set synthesis optimization goal to speed
# 2. ignore any LOCs in ngdbuild
# 3. perform timing-driven packing
# 4. use the highest par effort level
# 5. set the par extra effort level
# 6. pass "-instyle xflow" to the par command-line
# 7. generate a verbose report from trce
# 8. create the IEEE 1532 file during bitgen
project set "Optimization Goal" Speed
project set "Use LOC Constraints" false
project set "Perform Timing-Driven Packing and Placement" TRUE
project set "Place & Route Effort Level (Overall)" High
project set "Extra Effort (Highest (PAR level only)"  "Continue on
Impossible"
project set "Other Place & Route Command Line Options" "-intsyle xflow"
project set "Report Type" "Verbose Report"
project set "Create IEEE 1532 Configuration File" TRUE

# run the entire xst-to-trce flow
process run "Implement Design"

# close project
project close

# open project again
project open

# alter some partition properties
partition rerun /stopwatch/sixty implementation
partition rerun /stopwatch/lsbled synthesis
```

```
# rerun with only out-of-date partitions re-implemented
process run "Implement Design"

# close project
project close
```

Sample Tcl Script for Advanced Scripting

```
# create a new project and set device properties
project new watchvhd.ise
project set family virtex2p
project set device xc2VP2
project set package fG25
project get package
project set speed -7

# add the watch vhd files
xfile add dve_ccir_top.v
xfile add stopwatch.vhd statmatch.vhd cnt60.vhd dcm1.vhd decode.vhd
xfile add tenths.vhd hex2led.vhd watchvhd.ucf

# check that no partitions are present yet
set dus [search * -type partition]

# search for all design instances
set dus [search * -type instance]

# display all design instances
collection foreach du $dus {
puts "Name [object name $du], Type [object type $du]"
}

# confine search to the top-level instances only
set dus [search {/[^/]+/[^/]+} -type instance -regexp -exactmatch]

# define partitions for all the toplevel instances in the design (note
that the last partition created is returned)
partition new [search {/[^/]+/[^/]+} -type instance -regexp -
exactmatch]

# check that we created something
search * -type partition
```

```
# run with default values
process run "Implement Design"

# close project
project close
```

Chapter 4

PARTGen

The PARTGen program is compatible with the following Xilinx devices.

- Virtex™, Virtex™-E
- Virtex™-II
- Virtex™-II Pro, Virtex™-II Pro X
- Virtex™-4
- Virtex™-5 LX
- Spartan™-II, Spartan™-IIE
- Spartan™-3, Spartan™-3E, Spartan™-3L
- CoolRunner™ XPLA3, CoolRunner™-II
- XC9500™, XC9500XL™, XC9500XV™

This chapter describes PARTGen. The chapter contains the following sections.

PARTGen Overview

PARTGen is a command line tool that displays various levels of information about installed Xilinx devices and families. PARTGen does not require an input design file. It uses options specified on the command line to output architectural details.

PARTGen Syntax

Following is the command line syntax for PARTGen:

partgen *options*

options can be any number of the options listed in "PARTGen Options". They need not be listed in any order. Use spaces to separate multiple options.

Note: The command **partgen -i** lists output for every installed device.

PARTGen Input Files

PARTGen does not have any user input files.

PARTGen Output Files

PARTGen outputs two file types: partlist and PKG. Partlist and PKG files are produced with the -p and -v command line options. The -p option generates a terse version of the file, while the -v option generates a verbose version of the file.

Note: Partlist files are generated in both ASCII and XML formats.

Following are output file types produced by PARTGen:

- XCT file—Partlist file in ASCII format that contains detailed information about architectures and devices. See the "Partlist File" section for a detailed description of this file type.
- PKG files—ASCII formatted files that correlate IOBs with output pin names. PKG files are in XACT package format, which is a set of columns of information about the pins of a particular package. The -p option generates a three column entry describing the pins. The -v option adds five more columns of descriptive pin information.See the "PKG File" section for a detailed description of this file type.
- XML file—Partlist file in XML format.

PARTGen Options

This section describes the command line options and how they affect the behavior of PARTGen.

–arch (Print Information for Specified Architecture)

–arch *architecture_name*

The –arch option prints a list of devices, packages, and speeds for a specified architecture that has been installed.

Valid entries for *architecture_name* and the corresponding device product name are listed in the following table:

Table 4-1: **Values for** *architecture_name*

architecture_name	Corresponding Device Product Name
virtex	Virtex
virtex2	Virtex-II
virtex2p	Virtex-II Pro
virtexe	Virtex-E
virtex4	Virtex-4
virtex5	Virtex-5
spartan2	Spartan-II

Table 4-1: **Values for** *architecture_name*

architecture_name	Corresponding Device Product Name
spartan2e	Spartan-IIE
spartan3	Spartan-3
spartan3e	Spartan-3E
xc9500	XC9500
xc9500xl	XC9500XL
xc9500xv	XC9500XV
xpla3	CoolRunner XPLA3
xbr	CoolRunner-II

For example, entering the command **`partgen -arch virtex`** displays the following information:

```
Build:  /build/bcxfndry/HEAD/rtf
command:  partgen -arch virtex
Release 8.2i - PartGen HEAD
Copyright (c) 1995-2006 Xilinx, Inc.  All rights reserved.
v50             SPEEDS:         -6    -5    -4
        bg256
        cs144
        fg256
        pq240
        tq144
v100            SPEEDS:         -6    -5    -4
        bg256
        cs144
        fg256
        pq240
        tq144
v150            SPEEDS:         -6    -5    -4
        bg256
        bg352
        fg256
        fg456
        pq240
```

```
v200            SPEEDS:         -6    -5    -4
        bg256
        bg352
        fg256
        fg456
        pq240
v300            SPEEDS:         -6    -5    -4
        bg352
        bg432
        fg456
        pq240
v400            SPEEDS:         -6    -5    -4
        bg432
        bg560
        fg676
        hq240
v600            SPEEDS:         -6    -5    -4
        bg432
        bg560
        fg676
        fg680
        hq240
v800            SPEEDS:         -6    -5    -4
        bg432
        bg560
        fg676
        fg680
        hq240
v1000           SPEEDS:         -6    -5    -4
        bg560
        fg680
```

–i (Print a List of Devices, Packages, and Speeds)

The –i option prints out a list of devices, packages, and speeds that have been installed. Following is a portion of a sample display where the -i option has been used:

```
.
.
.
2s15            SPEEDS:         -6    -5    -5Q
        cs144
        tq144
        vq100
```

```
2s30            SPEEDS:         -6    -5    -5Q
        cs144
        tq144
        pq208
        vq100
2s50            SPEEDS:         -6    -5    -5Q
        tq144
        fg256
        pq208
2s100           SPEEDS:         -6    -5    -5Q
        tq144
        fg256
        fg456
        pq208
2s150           SPEEDS:         -6    -5    -5Q
        fg456
        fg256
        pq208
2s200           SPEEDS:         -6    -5    -5Q
        fg256
        fg456
        pq208
2s50e           SPEEDS:         -7    -6    -6Q
        ft256
        pq208
        tq144
2s100e          SPEEDS:         -7    -6    -6Q
        ft256
        fg456
        pq208
        tq144
2s150e          SPEEDS:         -7    -6    -6Q
        ft256
        fg456
        pq208
2s200e          SPEEDS:         -7    -6    -6Q
        ft256
        fg456
        pq208
2s300e          SPEEDS:         -7    -6    -6Q
        ft256
        fg456
        pq208
```

```
2s400e          SPEEDS:         -7    -6    -6Q
        ft256
        fg456
        fg676
2s600e          SPEEDS:         -7    -6    -6Q
        fg456
        fg676
3s50            SPEEDS:         -4
        pq208
        tq144
        vq100
3s200           SPEEDS:         -4
        ft256
        pq208
        tq144
        vq100
3s400           SPEEDS:         -4
        fg456
        ft256
        pq208
        tq144
3s1000          SPEEDS:         -4
        fg456
        fg676
        ft256
3s1500          SPEEDS:         -4
        fg456
        fg676
3s2000          SPEEDS:         -4
        fg676
        fg900
3s4000          SPEEDS:         -4
        fg900
        fg1156
3s5000          SPEEDS:         -4
        fg900
        fg1156
v50             SPEEDS:         -6    -5    -4
        bg256
        cs144
        fg256
        pq240
        tq144
```

–intstyle (Integration Style)

-intstyle {**ise** | **xflow** | **silent**}

The –intstyle option reduces screen output based on the integration style you are running. When using the –intstyle option, one of three modes must be specified: *ise*, *xflow*, or *silent*. The mode sets the way information is displayed in the following ways:

-intstyle ise

This mode indicates the program is being run as part of an integrated design environment.

-intstyle xflow

This mode indicates the program is being run as part of an integrated batch flow.

-intstyle silent

This mode limits screen output to warning and error messages only.

Note: The -intstyle option is automatically invoked when running in an integrated environment, such as Project Navigator or XFLOW.

–p (Creates Package file and Partlist Files)

–p *name*

The –p option generates a partlist.xct file for the specified name and also creates package files. All files are placed in the working directory. Valid name entries include architectures, devices, and parts. Following are example command line entries of each type:

–p virtex `(Architecture)`

–p xcv400 `(Device)`

–p v400bg432 `(Part)`

If an architecture, device, or part is not specified with the –p option, detailed information for every installed device is submitted to the partlist.xct file.

The –p option generates more detailed information than the -arch option, but less information than the –v option. The -p option generates a three column entry describing the pins. For each pin the following data appears:

- Column 1: contains either pin (user accessible pin) or pkgpin (dedicated pin).
- Column 2: specifies the pin name.
- Column 3: specifies the package pin.

For a description of the entries in the partlist file, see the "Partlist File".

–nopkgfile (No Package File)

The –nopkgfile option cancels the production of the .pkg files when the – p and –v options are used. The –nopkgfile option allows you to bypass creating .pkg files.

–v (Creates Package and Partlist Files)

–v *name*

The –v option generates a partlist file in ASCII and XML format for the specified name and also creates package files. Valid name entries include architectures, devices, parts. Following are example command line entries of each type:

–v virtex `(Architecture)`

–v xcv400 `(Device)`

–v v400bg432 `(Part)`

If no architecture, device, or part is specified with the –v option, information for every installed device is submitted to the partlist.xct file.

The –v option generates more detailed information than the –p option. The –p and –v options are mutually exclusive, that is, you can specify one or the other but not both. The -p option generates a three column entry describing the pins. The -v option adds six more columns of descriptive pin information.For each pin the following data appears:

- Column 1: contains either pin (user accessible pin) or pkgpin (dedicated pin).
- Column 2: specifies the pin name.
- Column 3: specifies the package pin.
- Column 4: IO_BANK is a positive integer associated with a VREF bank, or -1 to indicate no VREF bank association.
- Column 5: IO_BANK is a positive integer associated with a VCCO bank, or -1 to indicate no VCCO bank association.
- Column 6: specifies the function name, and consists of a string indicating how the pin is used. If the pin is dedicated, then the string will indicate the specific function. If the pin is a generic user pin, the string will be "IO." If the pin is multipurpose, an underscore-separated set of characters will make up the string.
- Column 7: indicates the closest CLB row/column to the pin.
- Column 8: indicates LVDS IOB association, consisting of an index (ranging from 0 to the number of LVDS pairs - 1) and the letter M or S. The value "N.A." indicates a non-LVDS pin.
- Column 9: indicates the flight time data (trace length) in units of microns. If no data is available, the column will contain zeros.

For a description of the entries in the partlist.xct file, see the "Partlist File" that follows.

Partlist File

The partlist file (XCT) contains detailed information about architectures and devices, including supported synthesis tools. Optionally, the partlist file can be generated in XML format with the

The partlist file is a series of part entries. There is one entry for every part supported in the installed software. The following subsections describe the information contained in the partlist.xct file.

Header

The first part is a header for the entry. The format of the entry looks like the following:

`part` *architecture family partname diename packagefilename*

Following is an example for the XCV50bg256:

```
partVIRTEX V50bg256 NA.die v50bg256.pkg
```

Device Attributes

The header is followed by a list of device attributes. Not all attributes are applicable to all devices.

- CLB row and column sizes: NCLBROWS=# NCLBCOLS=#
- Sub-family designation: STYLE=sub_family
  (For example, STYLE = Virtex2)
- Input registers: IN_FF_PER_IOB=#
- Output registers: OUT_FF_PER_IOB=#
- Number of pads per row and per column: NPADS_PER_ROW=#
  NPADS_PER_COL=#
- Bitstream information:
  - Number of frames: NFRAMES=#
  - Number bits/frame: NBITSPERFRAME=#

The preceding bulleted items display for both the -p and -v options. The following bulleted items are displayed only when using the -v option:

- Number of IOBS in device: NIOBS=#
- Number of bonded IOBS: NBIOBS=#
- Slices per CLB: SLICES_PER_CLB=#

  For slice-based architectures, such as Virtex.

  (For non-slice based architectures, assume one slice per CLB)
- Flip-flops for each slice: FFS_PER_SLICE=#
- Latches for each slice: CAN BE LATCHES={TRUE | FALSE}
- DLL blocks for Virtex, Virtex-E, Spartan-II, and Spartan-3 families, and DCMs for Virtex-II, Virtex-IIP, and Virtex-4 families that include the DLL functionality.
- LUTs in a slice: LUT_NAME=name LUT_SIZE=#
- Number of global buffers: NUM_GLOBAL_BUFFERS=#

  (The number of places where a buffer can drive a global clock combination.)

- External Clock IOB pins:
  - For Virtex, Virtex-E, Spartan-II, and Spartan-3E

    ```
    GCLKBUF0=PAD#, GCLKBUF1=PAD#,
    GCLKBUF2=PAD#, GCLKBUF3=PAD#
    ```

  - For Virtex-II, Virtex-II Pro, and Virtex-4:

    ```
    BUFGMUX0P=PAD#, BUFGMUX1P=PAD#,
    BUFGMUX2P=PAD#, BUFGMUX3P=PAD#, BUFGMUX4P=PAD#,
    BUFGMUX5P=PAD#,
    BUFGMUX6P=PAD#, BUFGMUX7P=PAD#
    ```

- Block RAM:

  ```
  NUM_BLK_RAMS=#
  BLK_RAM_COLS=# BLK_RAM_COL0=# BLK_RAMCOL1=# BLK_RAM_COL2=#
  BLK_RAM_COL_3=#
  BLK_RAM_SIZE=4096x1 BLK_RAM_SIZE=2048x2 BLK_RAM_SIZE=512x8
  BLK_RAM_SIZE=256x16
  ```

  Block RAM locations are given with reference to CLB columns. In the following example, Block RAM 5 is positioned in CLB column 32.

  ```
  NUM_BLK_RAMS=10 BLK_RAM_COL_5=32 BLK_RAM_SIZE=4096X1
  ```

- Select RAM:

  ```
  NUM_SEL_RAMS=#   SEL_RAM_SIZE=#X#
  ```

- Select Dual Port RAM:

  ```
  SEL_DP_RAM={TRUE|FALSE}
  ```

  This field indicates whether the select RAM can be used as a dual port ram. The assumption is that the number of addressable elements is reduced by half, that is, the size of the select RAM in Dual Port Mode is half that indicated by SEL_RAM_SIZE.

- Speed grade information: SPEEDGRADE=#
- Typical delay across a LUT for each speed grade: LUTDELAY=#
- Typical IOB input delay: IOB_IN_DELAY=#
- Typical IOB output delay: IOB_OUT_DELAY=#
- Maximum LUT constructed in a slice:

  ```
  MAX_LUT_PER_SLICE=#
  ```

  (From all the LUTs in the slice)

- Max LUT constructed in a CLB: MAX_LUT_PER_CLB=#

  This field describes how wide a LUT can be constructed in the CLB from the available LUTs in the slice.

- Number of internal 3-state buffers in a device: NUM_TBUFS PER ROW=#
- If unavailable on a particular device or package, PartGen reports:

  NUM_PPC=#

  NUM_GT=#

  NUM_MONITOR=#

  NUM_DPM=#

  NUM_PMCD=#

  NUM_DSP=#

  NUM_FIFO=#

  NUM_EMAC=#

  NUM_MULT=#

PKG File

The PKG files correlate IOBs with output pin names. The –p option generates a three column entry describing the pins. The –v option adds six more columns of descriptive pin information.

For example, the command **`partgen -p xc2v40`** generates the package files: 2v40cs144.pkg and 2v40fg256.pkg. Following is a portion of the package file for the 2v40cs144:

```
package 2v40cs144
pin PAD96 D3
pin PAD2 A3
pin PAD3 C4
pin PAD4 B4
.
.
.
```

The first column contains either *pin* (user accessible pin) or *pkgpin* (dedicated pin). The second column specifies the pin name. For user accessible pins, the name of the pin is the bonded pad name associated with an IOB on the device, or the name of a multi-purpose pin. For dedicated pins, the name is either the functional name of the pin, or no connection (N.C.). The third column specifies the package pin.

The command partgen –v generates package (.pkg) files and generates a nine column entry describing the pins. The first three columns are described in the preceding section.

The fourth and fifth columns, IO_BANK, is a positive integer associated with a bank, or –1 for no bank association. The sixth column, specifying function name, consists of a string indicating how the pin is used. If the pin is dedicated, then the string will indicate a specific function. If the pin is a generic user pin, the string is *IO*. If the pin is multipurpose, an underscore-separated set of characters will make up the string. The seventh column indicates the closest CLB row or column to the pin, and appears in the form R[0-9]C[0-9]. Column eight is comprised of a string for each pin associated with a LVDS IOB. The string consists of and index and the letter M or S. Index values will go from 0 to the number of LVDS pairs. The value for a non-LVDS pin will default to N.A. The ninth column is composed of flight-time data in units of microns. If no flight-time data is available, this column contains zeros.

Following are examples of the verbose pin descriptors in PARTGen:

```
package 2v40fg256
pkpin  N.C.  D9  -1 N.C.  N.A.  N.A.  0
pkpin  DONE  M12 -1 DONE  N.A.  N.A.  0
pkpin  VCCO  N1  -1 VCCO  N.A.  N.A.  0
.
.
.
```

Chapter 5

Logical Design Rule Check

This program is compatible with the following families:

- Virtex™, Virtex™-E
- Virtex™-II
- Virtex™-II Pro, Virtex™-II Pro X
- Virtex™-4
- Virtex™-5 LX
- Spartan™-II, Spartan™-IIE
- Spartan™-3, Spartan™-3E, Spartan™-3L
- CoolRunner™ XPLA3, CoolRunner™-II
- XC9500™, XC9500XL™, XC9500XV™

This chapter describes the Logical Design Rule Check (DRC). The chapter contains the following sections:

Logical DRC Overview

The Logical Design Rule Check (DRC), also known as the NGD DRC, comprises a series of tests to verify the logical design in the Native Generic Database (NGD) file. The Logical DRC performs device-independent checks.

The Logical DRC generates messages to show the status of the tests performed. Messages can be error messages (for conditions where the logic will not operate correctly) or warnings (for conditions where the logic is incomplete).

The Logical DRC runs automatically at the following times:

- At the end of NGDBuild, before NGDBuild writes out the NGD file

  NGDBuild writes out the NGD file if DRC warnings are discovered, but does not write out an NGD file if DRC errors are discovered.

- At the end of NetGen, before writing out the netlist file

  The netlist writer (NetGen) does not perform the entire DRC. It only performs the Net checks and Name checks. The netlist writer writes out a netlist file even if DRC warnings or errors are discovered.

Logical DRC Checks

The Logical DRC performs the following types of checks:

- Block check
- Net check
- Pad check
- Clock buffer check
- Name check
- Primitive pin check

The following sections describe these checks.

Block Check

The block check verifies that each terminal symbol in the NGD hierarchy (that is, each symbol that is not resolved to any lower-level components) is an NGD primitive. A block check failure is treated as an error. As part of the block check, the DRC also checks user-defined properties on symbols and the values on the properties to make sure they are legal.

Net Check

The net check determines the number of NGD primitive output pins (drivers), 3-state pins (drivers), and input pins (loads) on each signal in the design. If a signal does not have at least one driver (or one 3-state driver) and at least one load, a warning is generated. An error is generated if a signal has multiple non-3-state drivers or any combination of 3-state and non-3-state drivers. As part of the net check, the DRC also checks user-defined properties on signals and the values on the properties to make sure they are legal.

Pad Check

The pad check verifies that each signal connected to pad primitives obeys the following rules.

- If the PAD is an input pad, the signal to which it is connected can only be connected to the following types of primitives:
  - Buffers
  - Clock buffers
  - PULLUP
  - PULLDOWN
  - KEEPER
  - BSCAN

  The input signal can be attached to multiple primitives, but only one of each of the above types. For example, the signal can be connected to a buffer primitive, a clock buffer primitive, and a PULLUP primitive, but it cannot be connected to a buffer primitive and two clock buffer primitives. Also, the signal cannot be connected to both a PULLUP primitive and a PULLDOWN primitive. Any violation of the rules above results in an error, with the exception of signals attached to multiple pull-ups or pull-downs, which produces a warning. A signal that is not attached to any of the above types of primitives also produces a warning.

- If the PAD is an output pad, the signal it is attached to can only be connected to one of the following primitive outputs:
  - A single buffer primitive output
  - A single 3-state primitive output
  - A single BSCAN primitive

  In addition, the signal can also be connected to one of the following primitives:
  - A single PULLUP primitive
  - A single PULLDOWN primitive
  - A single KEEPER primitive

  Any other primitive output connections on the signal results in an error.

  If the condition above is met, the output PAD signal may also be connected to one clock buffer primitive *input*, one buffer primitive *input*, or both.
- If the PAD is a bidirectional or unbonded pad, the signal it is attached to must obey the rules stated above for input and output pads. Any other primitive connections on the signal results in an error. The signal connected to the pad must be configured as both an input and an output signal; if it is not, you receive a warning.
- If the signal attached to the pad has a connection to a top-level symbol of the design, that top-level symbol pin must have the same type as the pad pin, except that output pads can be associated with 3-state top-level pins. A violation of this rule results in a warning.
- If a signal is connected to multiple pads, an error is generated. If a signal is connected to multiple top-level pins, a warning is generated.

Clock Buffer Check

The clock buffer configuration check verifies that the output of each clock buffer primitive is connected to only inverter, flip-flop or latch primitive clock inputs, or other clock buffer inputs. Violations are treated as warnings.

Name Check

The name check verifies the uniqueness of names on NGD objects using the following criteria:

- Pin names must be unique within a symbol. A violation results in an error.
- Instance names must be unique within the instance's position in the hierarchy (that is, a symbol cannot have two symbols with the same name under it). A violation results in a warning.
- Signal names must be unique within the signal's hierarchical level (that is, if you push down into a symbol, you cannot have two signals with the same name). A violation results in a warning.
- Global signal names must be unique within the design. A violation results in a warning.

Primitive Pin Check

The primitive pin check verifies that certain pins on certain primitives are connected to signals in the design. The following table shows which pins are tested on each NGD primitive type.

Table 5-1: **Checked Primitive Pins**

NGD Primitive	Pins Checked
X_MUX	SEL
X_TRI	IN, OUT, and CTL
X_FF	IN, OUT, and CLK
X_LATCH	IN, OUT, and CLK
X_IPAD	PAD
X_OPAD	PAD
X_BPAD	PAD

Note: If one of these pins is not connected to a signal, you receive a warning.

Chapter 6

NGDBuild

This program is compatible with the following families:

- Virtex™, Virtex™-E
- Virtex™-II
- Virtex™-II Pro, Virtex™-II Pro X
- Virtex™-4
- Virtex™-5 LX
- Spartan™-II, Spartan™-IIE
- Spartan™-3, Spartan™-3E, Spartan™-3L
- CoolRunner™ XPLA3, CoolRunner™-II
- XC9500™, XC9500XL™, XC9500XV™

This chapter describes the NGDBuild program. The chapter contains the following sections

NGDBuild Overview

NGDBuild reads in a netlist file in EDIF or NGC format and creates a Xilinx Native Generic Database (NGD) file that contains a logical description of the design in terms of logic elements, such as AND gates, OR gates, decoders, flip-flops, and RAMs.

The NGD file contains both a logical description of the design reduced to Xilinx Native Generic Database (NGD) primitives and a description of the original hierarchy expressed in the input netlist. The output NGD file can be mapped to the desired device family.

The following figure shows a simplified version of the NGDBuild design flow. NGDBuild invokes other programs that are not shown in the following figure.

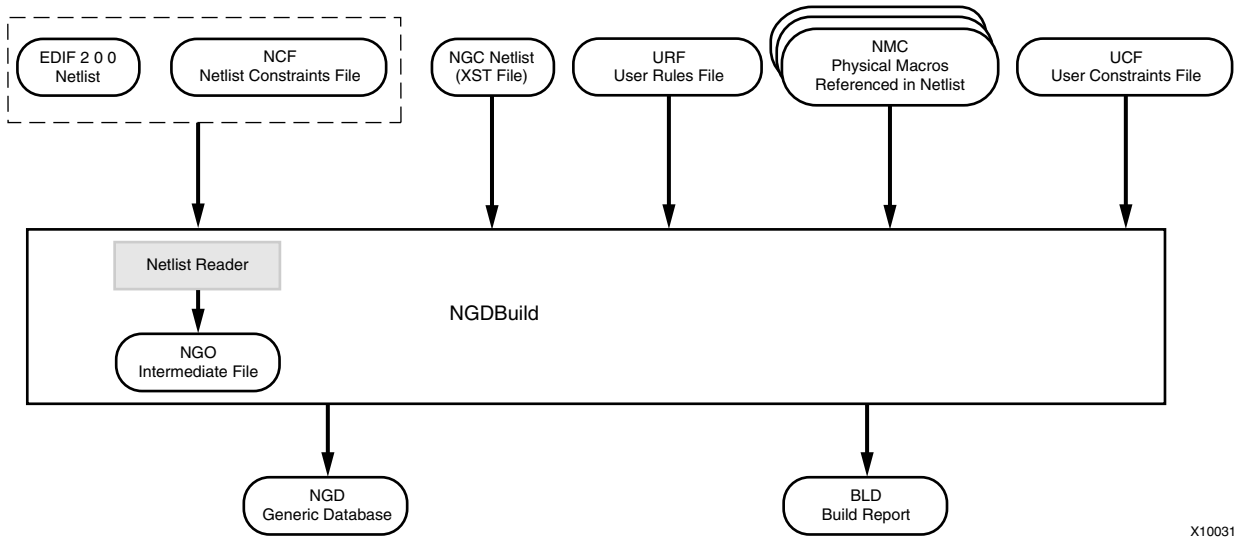


Figure 6-1: **NGDBuild Design Flow**

Converting a Netlist to an NGD File

NGDBuild performs the following steps to convert a netlist to an NGD file:

1. Reads the source netlist

   NGDBuild invokes the Netlist Launcher. The Netlist Launcher determines the input netlist type and starts the appropriate netlist reader program. The netlist reader incorporates NCF files associated with each netlist. NCF files contain timing and layout constraints for each module. The Netlist Launcher is described in detail in the "Netlist Launcher (Netlister)" in Appendix B.

2. Reduces all components in the design to NGD primitives

   NGDBuild merges components that reference other files. NGDBuild also finds the appropriate system library components, physical macros (NMC files), and behavioral models.

3. Checks the design by running a Logical Design Rule Check (DRC) on the converted design

   Logical DRC is a series of tests on a logical design. It is described in Chapter 5, "Logical Design Rule Check".

4. Writes an NGD file as output

Note: This procedure, the Netlist Launcher, and the netlist reader programs are described in more detail in Appendix B, "EDIF2NGD, and NGDBuild".

NGDBuild Syntax

The following command reads the design into the Xilinx Development system and converts it to an NGD file:

ngdbuild [*options*] *design_name* [*ngd_file*[.ngd]]

options can be any number of the NGDBuild command line switches listed in "NGDBuild Options". They can be listed in any order. Separate multiple options with spaces.

design_name is the top-level name of the design file you want to process. To ensure the design processes correctly, specify a file extension for the input file, using one of the legal file extensions specified in "NGDBuild Input Files". Using an incorrect or nonexistent file extension causes NGDBuild to fail without creating an NGD file. If you use an incorrect file extension, NGDBuild may issue an "unexpanded" error.

Note: If you are using an NGC file as your input design, it is recommended that you specify the .ngc extension. If NGDBuild finds an EDIF netlist or NGO file in the project directory, it does not check for an NGC file.

ngd_file[**.ngd**] is the output file in NGD format. The output file name, its extension, and its location are determined as follows:

- If you do not specify an output file name, the output file has the same name as the input file, with an .ngd extension.
- If you specify an output file name with no extension, NGDBuild appends the .ngd extension to the file name.
- If you specify a file name with an extension other than .ngd, you get an error message and NGDBuild does not run.
- If the output file already exists, it is overwritten with the new file.

NGDBuild Input Files

NGDBuild uses the following files as input:

- Design file—The input design can be an EDIF 2 0 0 or NGC netlist file. If the input netlist is in another format that the Netlist Launcher recognizes, the Netlist Launcher invokes the program necessary to convert the netlist to EDIF format, then invokes the appropriate netlist reader, EDIF2NGD.

  With the default Netlist Launcher options, NGDBuild recognizes and processes files with the extensions shown in the following table. NGDBuild searches the top-level design netlist directory for a netlist file with one of the extensions. By default, NGDBuild searches for an EDIF file first.

File Type	Recognized Extensions
EDIF	.sedif, .edn, .edf, .edif
NGC	.ngc

Note: Remove all out of date netlist files from your directory. Obsolete netlist files may cause errors in NGDBuild.

- UCF file—The User Constraints File is an ASCII file that you create. You can create this file by hand or by using the Constraints Editor. See the online Help provided with the Constraints Editor for more information. The UCF file contains timing and layout constraints that affect how the logical design is implemented in the target device. The constraints in the file are added to the information in the output NGD file. For detailed information on constraints, see the *Constraints Guide*.

  By default, NGDBuild reads the constraints in the UCF file automatically if the UCF file has the same base name as the input design file and a .ucf extension. You can override the default behavior and specify a different constraints file with the –uc option. See "–uc (User Constraints File)" for more information.

  Note: NGDBuild allows one UCF file as input.

- NCF —The netlist constraints file is produced by a CAE vendor toolset. This file contains constraints specified within the toolset. The netlist reader invoked by NGDBuild reads the constraints in this file if the NCF has the same name as the input EDIF netlist. It adds the constraints to the intermediate NGO file and the output NGD file. NCF files do no bind to NGC files because they are read in and annotated to the NGO file during an **`edif2ngd`** conversion. This also implies that unlike UCF files, NCF constraints only bind to a single edif netlist; they do not cross file hierarchies.

  Note: NGDBuild checks to make sure the NGO file is up-to-date and reruns **`edif2ngd`** only when the EDIF has a timestamp that is newer than the NGO file. Updating the NCF has no affect on whether **`edif2ngd`** is rerun. Therefore, if the NGO is up-to-date and you only update the NCF file (not the EDIF), use the **`-nt on`** option to force the regeneration of the NGO file from the unchanged EDIF and new NCF. See "–nt (Netlist Translation Type)" for more information.

- URF file — The User Rules File (URF) is an ASCII file that you create. The Netlist Launcher reads this file to determine the acceptable netlist input files, the netlist readers that read these files, and the default netlist reader options. This file also allows you to specify third-party tool commands for processing designs. The URF can add to or override the rules in the system rules file.

  You can specify the location of the user rules file with the NGDBuild –ur option. The user rules file must have a .urf extension. See "–ur (Read User Rules File)" or "User Rules File" in Appendix B for more information.

- NGC file—This binary file can be used as a top-level design file or as a module file:

  - Top-level design file

    This file is output by the Xilinx Synthesis Technology (XST) tool. See the description of design files earlier in this section for details.

    Note: This is not a "true" netlist file. However, it is referred to as a netlist in this context to differentiate it from the NGC module file. NGC files are equivalent to NGO files created by edif2ngd, but are created by other Xilinx applications: XST and CORE Generator.

- NMC files—These physical macros are binary files that contain the implementation of a physical macro instantiated in the design. NGDBuild reads the NMC file to create a functional simulation model for the macro.

Unless a full path is provided to NGDBuild, it searches for netlist, NGC, NMC, and MEM files in the following locations:

- The working directory from which NGDBuild was invoked.
- The path specified for the top-level design netlist on the NGDBuild command line.
- Any path specified with the "–sd (Search Specified Directory)" on the NGDBuild command line.

NGDBuild Output Files

NGDBuild creates the following files as output:

- NGD file—This binary file contains a logical description of the design in terms of both its original components and hierarchy and the NGD primitives to which the design is reduced.
- BLD file—This build report file contains information about the NGDBuild run and about the subprocesses run by NGDBuild. Subprocesses include EDIF2NGD, and programs specified in the URF. The BLD file has the same root name as the output NGD file and a .bld extension. The file is written into the same directory as the output NGD file.

NGDBuild Intermediate Files

NGO files—These binary files contain a logical description of the design in terms of its original components and hierarchy. These files are created when NGDBuild reads the input EDIF netlist. If these files already exist, NGDBuild reads the existing files. If these files do not exist or are out of date, NGDBuild creates them.

NGDBuild Options

This section describes the NGDBuild command line options.

–a (Add PADs to Top-Level Port Signals)

If the top-level input netlist is in EDIF format, the –a option causes NGDBuild to add a PAD symbol to every signal that is connected to a port on the root-level cell. This option has no effect on lower-level netlists.

Using the –a option depends on the behavior of your third-party EDIF writer. If your EDIF writer treats pads as instances (like other library components), do not use –a. If your EDIF writer treats pads as hierarchical ports, use –a to infer actual pad symbols. If you do not use –a where necessary, logic may be improperly removed during mapping.

For EDIF files produced by Mentor Graphics and Cadence schematic tools, the –a option is set automatically; you do *not* have to enter –a explicitly for these vendors.

Note: The NGDBuild –a option corresponds to the EDIF2NGD –a option. If you run EDIF2NGD on the top-level EDIF netlist separately, rather than allowing NGDBuild to run EDIF2NGD, you must use the two –a options consistently. If you previously ran NGDBuild on your design and NGO files are present, you must use the –nt on option the first time you use –a. This forces a rebuild of the NGO files, allowing EDIF2NGD to run the –a option.

–aul (Allow Unmatched LOCs)

By default (without the –aul option), NGDBuild generates an error if the constraints specified for pin, net, or instance names in the UCF or NCF file cannot be found in the design. If this error occurs, an NGD file is not written. If you enter the –aul option, NGDBuild generates a warning instead of an error for LOC constraints and writes an NGD file.

You may want to run NGDBuild with the –aul option if your constraints file includes location constraints for pin, net, or instance names that have not yet been defined in the HDL or schematic. This allows you to maintain one version of your constraints files for both partially complete and final designs.

Note: When using this option, make sure you do not have misspelled net or instance names in your design. Misspelled names may cause inaccurate placing and routing.

–bm (Specify BMM Files)

-bm *file_name* [.bmm]

The –bm option specifies a switch for the .bmm files. If the file extension is missing, a .bmm file extension is assumed. If this option is unspecified, the ELF or MEM root file name with a .bmm extension is assumed. If only this option is given, then Ngdbuild verifies that the .bmm file is syntactically correct and makes sure that the instances specified in the .bmm file exist in the design. Only one –bm option can be used

–dd (Destination Directory)

–dd *NGOoutput_directory*

The –dd option specifies the directory for intermediate files (design NGO files and netlist files). If the –dd option is not specified, files are placed in the current directory.

–f (Execute Commands File)

–f *command_file*

The –f option executes the command line arguments in the specified *command_file*. For more information on the –f option, see "–f (Execute Commands File)" in Chapter 1.

–i (Ignore UCF File)

By default (without the –i option), NGDBuild reads the constraints in the UCF file automatically if the UCF file in the top-level design netlist directory has the same base name as the input design file and a .ucf extension. The –i option ignores the UCF file.

Note: If you use this option, do not use the –uc option.

–insert_keep_hierarchy

This option automatically attaches the KEEP_HIERARCHY constraint to each input netlist. It should only be used when performing a bottom-up synthesis flow, where separate netlists are created for each piece of hierarchy. It is highly recommended that when using this option, good design practices are used as described in the *Synthesis and Simulation Design Guide*.

Note: Care should be taken when trying to use this option with Cores, as they may not be coded for maintaining hierarchy.

–intstyle (Integration Style)

-intstyle {**ise** | **xflow** | **silent**}

The –intstyle option reduces screen output based on the integration style you are running. When using the –intstyle option, one of three modes must be specified: *ise*, *xflow*, or *silent*. The mode sets the way information is displayed in the following ways:

-intstyle ise

This mode indicates the program is being run as part of an integrated design environment.

-intstyle xflow

This mode indicates the program is being run as part of an integrated batch flow.

-intstyle silent

This mode limits screen output to warning and error messages only.

Note: The -intstyle option is automatically invoked when running in an integrated environment, such as Project Navigator or XFLOW.

–l (Libraries to Search)

-l *libname*

The –l option indicates the list of libraries to search when determining what library components were used to build the design. This option is passed to the appropriate netlist reader. The information allows NGDBuild to determine the source of the design's components so it can resolve the components to NGD primitives.

You can specify multiple libraries by entering multiple **–l** *libname* entries on the NGDBuild command line.

The allowable entries for *libname* are the following:

- **xilinxun** (Xilinx Unified library)
- **synopsys**

Note: You do not have to enter xilinxun with a –l option. The Xilinx Development System tools automatically access these libraries. In cases where NGDBuild automatically detects Synopsys designs (for example, the netlist extension is .sedif), you do not have to enter synopsys with a –l option.

–modular assemble (Module Assembly)

–modular assemble -pimpath *pim_directory_path*
–use_pim *module_name1* **–use_pim** *module_name2 ...*

Note: This option supports current FPGA device families.

The –modular assemble option starts the final phase of the Modular Design flow. In this "Final Assembly" phase, the team leader uses this option to create a fully expanded NGD file that contains logic from the top-level design and each of the Physically Implemented Modules (PIMs). The team leader then implements this NGD file.

Run this option from the top-level design directory.

If you are running the standard Modular Design flow, you do not need to use the –pimpath option. If you do not use the –use_pim option, NGDBuild searches the PIM directory's subdirectories for NGO files with names that match their subdirectory. It assembles the final design using these NGO files.

If you are running Modular Design in a Partial Assembly flow, use the –pimpath option to specify the directory that contains the PIMs. Use the –use_pim option to identify all the modules in the PIM directory that have been published. Be sure to use exact names of the PIMs, including the proper spelling and capitalization. The input design file should be the NGO file of the top-level design.

Note: When running Modular Design in a Partial Assembly flow, you must use the –modular assemble option with the –u option.

–modular initial (Initial Budgeting of Modular Design)

Note: This option supports current FPGA devices only.

The –modular initial option starts the first phase of the Modular Design flow. In this "Initial Budgeting" phase, the team leader uses this option to generate an NGO and NGD file for the top-level design with all of the instantiated modules represented as unexpanded blocks. After running this option, the team leader sets up initial budgeting for the design. This includes assigning top-level timing constraints and location constraints for various resources, including each module, using the Floorplanner and Constraints Editor tools.

Note: You cannot use the NGD file for mapping.

Run this option from the top-level design directory. The input design file should be an EDIF netlist or an NGC netlist from XST.

–modular module (Active Module Implementation)

–modular module -active *module_name*

Note: This option supports current FPGA devices. You *cannot* use NCD files from previous software releases with Modular Design in this release. You must generate new NCD files with the current release of the software.

The –modular module option starts the second phase of the Modular Design flow. In this "Active Module Implementation" phase, each team member creates an NGD file with just the specified "active" module expanded. This NGD file is named after the top-level design.

Run this option from the active module directory. This directory should include the active module netlist file and the top-level UCF file generated during the Initial Budgeting phase. You must specify the name of the active module after the –active option, and use the top-level NGO file as the input design file.

After running this option, you can then run MAP and PAR to create a Physically Implemented Module (PIM). Then, you must run PIMCreate to publish the PIM to the PIMs directory. PIMCreate copies the local, implemented module file, including the NGO, NGM and NCD files, to the appropriate module directory inside the PIMs directory and renames the files to *module_name.extension*.

To run PIMCreate, type the following on the command line:

pimcreate *pim_directory* **-ncd** *design_name*_**routed.ncd**

Note: When running Modular Design in an Incremental Guide flow, run NGDBuild with the –pimpath and –use_pim options normally reserved for the –modular assemble option.

–nt (Netlist Translation Type)

–nt {**timestamp** | **on** | **off**}

The –nt option determines how timestamps are treated by the Netlist Launcher when it is invoked by NGDBuild. A timestamp is information in a file that indicates the date and time the file was created. The timestamp option (which is the default if no –nt option is specified) instructs the Netlist Launcher to perform the normal timestamp check and update NGO files according to their timestamps. The on option translates netlists regardless of timestamps (rebuilding all NGO files), and the off option does not rebuild an existing NGO file, regardless of its timestamp.

–p (Part Number)

–p *part*

The –p option specifies the part into which the design is implemented. The –p option can specify an architecture only, a complete part specification (device, package, and speed), or a partial specification (for example, device and package only).

The syntax for the –p option is described in "–p (Part Number)" in Chapter 1. Examples of part entries are **XCV50-TQ144** and **XCV50-TQ144-5**.

When you specify the *part*, the NGD file produced by NGDBuild is optimized for mapping into that architecture.

You do not have to specify a –p option if your NGO file already contains information about the desired vendor and family (for example, if you placed a PART property in a schematic or a CONFIG PART statement in a UCF file). However, you can override the information in the NGO file with the –p option when you run NGDBuild.

Note: If you are running the Modular Design flow and are targeting a part different from the one specified in your source design, you must specify the part type using the **-p** option *every time* you run NGDBuild.

–r (Ignore LOC Constraints)

The –r option eliminates all location constraints (LOC=) found in the input netlist or UCF file. Use this option when you migrate to a different device or architecture, because locations in one architecture may not match locations in another.

–sd (Search Specified Directory)

–sd *search_path*

The –sd option adds the specified *search_path* to the list of directories to search when resolving file references (that is, files specified in the schematic with a FILE=*filename* property) and when searching for netlist, NGO, NGC, NMC, and MEM files. You do not have to specify a search path for the top-level design netlist directory, because it is automatically searched by NGDBuild.

The *search_path* must be separated from the –sd by spaces or tabs (for example, **–sd designs** is correct, **–sddesigns** is not). You can specify multiple –sd options on the command line. Each must be preceded with –sd; you cannot combine multiple search_path specifiers after one –sd. For example, the following syntax is *not* acceptable.

```
-sd /home/macros/counter /home/designs/pal2
```

The following syntax is acceptable.

```
-sd /home/macros/counter -sd /home/designs/pal2
```

–u (Allow Unexpanded Blocks)

In the default behavior of NGDBuild (without the –u option), NGDBuild generates an error if a block in the design cannot be expanded to NGD primitives. If this error occurs, an NGD file is not written. If you enter the –u option, NGDBuild generates a warning instead of an error if a block cannot be expanded, and writes an NGD file containing the unexpanded block.

You may want to run NGDBuild with the –u option to perform preliminary mapping, placement and routing, timing analysis, or simulation on the design even though the design is not complete. To ensure the unexpanded blocks remains in the design when it is mapped, run the MAP program with the –u (Do Not Remove Unused Logic) option, as described in "–u (Do Not Remove Unused Logic)" in Chapter 7.

–uc (User Constraints File)

–uc *ucf_file* [**.ucf**]

The –uc option specifies a User Constraints File (UCF) for the Netlist Launcher to read. The UCF file contains timing and layout constraints that affect the way the logical design is implemented in the target device.

The user constraints file must have a .ucf extension. If you specify a user constraints file without an extension, NGDBuild appends the .ucf extension to the file name. If you specify a file name with an extension other than .ucf, you get an error message and NGDBuild does not run.

If you do not enter a –uc option and a UCF file exists with the same base name as the input design file and a .ucf extension, NGDBuild automatically reads the constraints in this UCF file.

See the *Constraints Guide* for more information on the UCF file.

Note: NGDBuild only allows one UCF file as input. Therefore, you cannot specify multiple –uc options on the command line. Also, if you use this option, do not use the –i option.

–ur (Read User Rules File)

–ur *rules_file* [**.urf**]

The –ur option specifies a user rules file for the Netlist Launcher to access. This file determines the acceptable netlist input files, the netlist readers that read these files, and the default netlist reader options. This file also allows you to specify third-party tool commands for processing designs.

The user rules file must have a .urf extension. If you specify a user rules file with no extension, NGDBuild appends the .urf extension to the file name. If you specify a file name with an extension other than .urf, you get an error message and NGDBuild does not run.

See "User Rules File" in Appendix B for more information.

–verbose (Report All Messages)

The –verbose option enhances NGDBuild screen output to include all messages output by the tools run: NGDBuild, the netlist launcher, and the netlist reader. This option is useful if you want to review details about the tools run.

Chapter 7

MAP

This program is compatible with the following families:

- Virtex™, Virtex™-E
- Virtex™-II
- Virtex™-II Pro, Virtex™-II Pro X
- Virtex™-4
- Virtex™-5 LX
- Spartan™-II, Spartan™-IIE
- Spartan™-3, Spartan™-3E, Spartan™-3L

This chapter describes the MAP program, which is used during the implementation process to map a logical design to a Xilinx FPGA. This chapter contains the following sections:

MAP Overview

The MAP program maps a logical design to a Xilinx FPGA. The input to MAP is an NGD file, which is generated using the NGDBuild program. The NGD file contains a logical description of the design that includes both the hierarchical components used to develop the design and the lower level Xilinx primitives. The NGD file also contains any number of NMC (macro library) files, each of which contains the definition of a physical macro.

MAP first performs a logical DRC (Design Rule Check) on the design in the NGD file. MAP then maps the design logic to the components (logic cells, I/O cells, and other components) in the target Xilinx FPGA.

The output from MAP is an NCD (Native Circuit Description) file—a physical representation of the design mapped to the components in the targeted Xilinx FPGA. The mapped NCD file can then be placed and routed using the PAR program.

The following figure shows the MAP design flow:

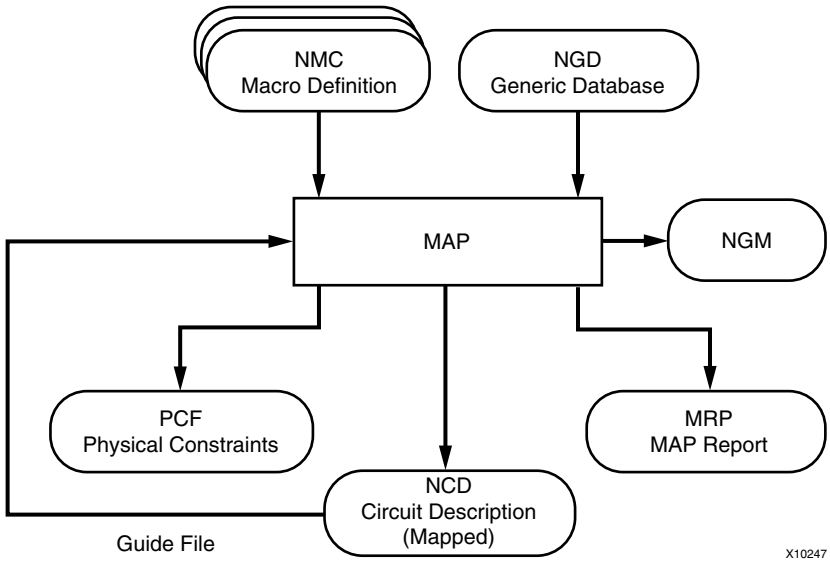


Figure 7-1: **MAP design flow**

MAP Syntax

The following syntax maps your logical design:

map [*options*] *infile*[**.ngd**] [*pcf_file*[**.pcf**]]

options can be any number of the MAP command line options listed in the "MAP Options" section of this chapter. They do not need to be listed in any particular order. Separate multiple options with spaces.

infile[**.ngd**] is the input NGD file. You do not have to enter the .ngd extension, since map looks for an NGD file as input.

pcf_file[**.pcf**] is the name of the output physical constraints file (PCF). Specifying a physical constraints file name is optional, and you do not have to enter the .pcf extension. If not specified, the physical constraints file name and its location are determined in the following ways:

- If you do not specify a physical constraints file name on the command line, the physical constraints file has the same name as the output file, with a .pcf extension. The file is placed in the output file's directory.
- If you specify a physical constraints file with no path specifier (for example, **cpu_1.pcf** instead of **/home/designs/cpu_1.pcf**), the .pcf file is placed in the current working directory.
- If you specify a physical constraints file name with a full path specifier (for example, **/home/designs/cpu_1.pcf**), the physical constraints file is placed in the specified directory.

- If the physical constraints file already exists, MAP reads the file, checks it for syntax errors, and overwrites the schematic-generated section of the file. MAP also checks the user-generated section for errors and corrects errors by commenting out physical constraints in the file or by halting the operation. If no errors are found in the user-generated section, the section is unchanged.

Note: For a discussion of the output file name and its location, see "–o (Output File Name)".

MAP Input Files

MAP uses the following files as input:

- NGD file—Native Generic Database file. This file contains a logical description of the design expressed both in terms of the hierarchy used when the design was first created and in terms of lower-level Xilinx primitives to which the hierarchy resolves. The file also contains all of the constraints applied to the design during design entry or entered in a UCF (User Constraints File). The NGD file is created by the NGDBuild program.
- NMC file—Macro library file. An NMC file contains the definition of a physical macro. When there are macro instances in the NGD design file, NMC files are used to define the macro instances. There is one NMC file for each *type* of macro in the design file.
- Guide NCD file—An optional input file generated from a previous MAP run. An NCD file contains a physical description of the design in terms of the components in the target Xilinx device. A guide NCD file is an output NCD file from a previous MAP run that is used as an input to guide a later MAP run.
- Guide NGM file—An optional input file, which is a binary design file containing all of the data in the input NGD file as well as information on the physical design produced by the mapping. See "Guided Mapping" for details.

MAP Output Files

Output from MAP consists of the following files:

- NCD (Native Circuit Description) file—a physical description of the design in terms of the components in the target Xilinx device. For a discussion of the output NCD file name and its location, see "–o (Output File Name)".
- PCF (Physical Constraints File)—an ASCII text file that contains constraints specified during design entry expressed in terms of physical elements. The physical constraints in the PCF are expressed in Xilinx's constraint language.

  MAP creates a PCF file if one does not exist or rewrites an existing file by overwriting the schematic-generated section of the file (between the statements SCHEMATIC START and SCHEMATIC END). For an existing physical constraints file, MAP also checks the user-generated section for syntax errors and signals errors by halting the operation. If no errors are found in the user-generated section, the section is unchanged.
- NGM file—a binary design file that contains all of the data in the input NGD file as well as information on the physical design produced by mapping. The NGM file is used to correlate the back-annotated design netlist to the structure and naming of the source design.

- MRP (MAP report)—a file that contains information about the MAP run. The MRP file lists any errors and warnings found in the design, lists design attributes specified, and details on how the design was mapped (for example, the logic that was removed or added and how signals and symbols in the logical design were mapped into signals and components in the physical design). The file also supplies statistics about component usage in the mapped design. See "MAP Report (MRP) File" for more details.

The MRP, PCF, and NGM files produced by a MAP run all have the same name as the output NCD file, with the appropriate extension. If the MRP, PCF, or NGM files already exist, they are overwritten by the new files.

MAP Options

The following table summarizes the MAP command line options and the supported architectures for each option.

Table 7-1: **Map Options and Architectures**

Options	Architectures
–bp	All FPGA architectures
–c	All FPGA architectures
–cm	All FPGA architectures
–detail	All FPGA architectures
–equivalent_register_removal	Virtex-4 architectures
–f	All FPGA architectures
–gf	All FPGA architectures
–global_opt	Virtex-4 architectures
–gm	All FPGA architectures
–ignore_keep_hierarchy	All FPGA architectures
–intstyle	All FPGA architectures
–ir	All FPGA architectures
–ise	All FPGA architectures
–k	All FPGA architectures
–l	All FPGA architectures
–o	All FPGA architectures
–ol	All FPGA architectures
–p	All FPGA architectures
–pr	All FPGA architectures
–r	All FPGA architectures
–register_duplication	All FPGA architectures

Table 7-1: **Map Options and Architectures**

Options	Architectures
–retiming	Virtex-4 architectures
–t	All FPGA architectures
–timing	All FPGA architectures
–tx	Not used for Virtex-4, Spartan-3, Spartan-3E, or Spartan-3L architectures
–u	All FPGA architectures
–xe	All FPGA architectures

–bp (Map Slice Logic)

The block RAM mapping option is enabled when the –bp option is specified. When block RAM mapping is enabled, MAP attempts to place LUTs and FFs into single-output, single-port block RAMs.

You can create a file containing a list of register output nets that you want converted into block RAM outputs. To instruct MAP to use this file, set the environment variable XIL_MAP_BRAM_FILE to the file name. MAP looks for this environment variable when the –bp option is specified. Only those output nets listed in the file are made into block RAM outputs.

Note: Because block RAM outputs are synchronous and can only be reset, the registers packed into a block RAM must also be synchronous reset.

–c (Pack CLBs)

–c [*packfactor*]

The –c option determines the degree to which CLBs are packed when the design is mapped. The valid range of values for the *packfactor* is 0–100.

The *packfactor* values ranging from 1 to 100 roughly specify the percentage of CLBs available in a target device for packing your design's logic.

A *packfactor* of 100 means that all CLBs in a target part are available for design logic. A *packfactor* of 100 results in minimum packing density, while a *packfactor* of 1 represents maximum packing density. Specifying a lower *packfactor* results in a denser design, but the design may then be more difficult to place and route.

The **–c 0** option specifies that only *related* logic (that is, logic having signals in common) should be packed into a single CLB. Specifying **–c 0** yields the least densely packed design.

For values of –c from 1 to 100, MAP merges unrelated logic into the same CLB only if the design requires more resources than are available in the target device (an *overmapped* design). If there are *more* resources available in the target device than are needed by your design, the number of CLBs utilized when **–c 100** is specified may equal the number required when **–c 0** is specified.

Note: The **–c 1** setting should only be used to determine the maximum density (minimum area) to which a design can be packed. Xilinx does not recommend using this option in the actual implementation of your design. Designs packed to this maximum density generally have longer run times, severe routing congestion problems in PAR, and poor design performance.

The default *packfactor* value is 100%. This value applies if you do not specify the –c option, or enter the –c option without a *packfactor* value.

Processing a design with the **–c 0** option is a good way to get a first estimate of the number of CLBs required by your design.

–cm (Cover Mode)

–cm {**area** | **speed** | **balanced**}

The –cm option specifies the criteria used during the *cover* phase of MAP. In this phase, MAP assigns the logic to CLB function generators (LUTs). Use the area, speed, and balanced settings as follows:

- The **area** setting makes reducing the number of LUTs (and therefore the number of CLBs) the highest priority.
- The behavior of the **speed** setting depends on the existence of user-specified timing constraints. For the design with user-specified timing constraints, the speed mode makes achieving timing constraints the highest priority and reducing the number of levels of LUTS (the number of LUTs a path passes through) the next priority. For the design with no user-specified timing constraints, the speed mode makes achieving maximum system frequency the highest priority and reducing the number levels of LUTs the next priority. This setting makes it easiest to achieve timing constraints after the design is placed and routed. For most designs, there is a small increase in the number of LUTs (compared to the area setting), but in some cases the increase may be large.
- The **balanced** setting balances the two priorities – achieving timing requirements and reducing the number of LUTs. It produces results similar to the speed setting but avoids the possibility of a large increase in the number of LUTs. For a design with user-specified timing constraints, the balanced mode makes achieving timing constraints the highest priority and reducing the number of LUTS the next priority. For the design with no user-specified timing constraints, the balanced mode makes achieving maximum system frequency the highest priority and reducing the number of LUTs the next priority.

The default setting for the –cm option is **area** (cover for minimum number of LUTs).

–detail (Write Out Detailed MAP Report)

This option writes out a detailed MAP report. The option replaces the MAP_REPORT_DETAIL environment variable.

–equivalent_register_removal (Remove Redundant Registers)

-equivalent_register_removal *on|off*

With this option *on*, any registers with redundant functionality are examined to see if their removal will increase clock frequencies. This option is available for Virtex-4 designs. By default, this option is *on*.

Note: This option is available only when "–global_opt (Global Optimization)" is used.

–f (Execute Commands File)

-f *command_file*

The –f option executes the command line arguments in the specified *command_file*. For more information on the –f option, see "–f (Execute Commands File)" in Chapter 1.

–gf (Guide NCD File)

-gf *guidefile*

The –gf option specifies the name of an existing NCD file (from a previous MAP run) to be used as a guide for the current MAP run. Guided mapping also uses an NGM file. For a description of guided mapping, see the "Guided Mapping" section of this chapter.

Note: When using guided mapping with the –timing (Timing-Driven Packing and Placement), Xilinx recommends using a placed NCD as the guide file. A placed NCD is produced by running MAP –timing or PAR.

–global_opt (Global Optimization)

-global_opt *on|off*

This option directs MAP to perform global optimization routines on the fully assembled netlist before mapping the design. Global optimization includes logic remapping and trimming, logic and register replication and optimization, and logic replacement of tristates. These routines will extend the runtime of Map because extra processing occurs. This option is available for Virtex-4 designs. By default this option is off.

When using the –global_opt option, Modular and Incremental design flows are not recommended. Incremental guide mode is explicitly prohibited and Formal Verification flows will be negatively affected. Also, certain MAP properties are not allowed in conjunction with global optimization, such as the –gf, –ol, and –u options.

Note: See also the "–equivalent_register_removal (Remove Redundant Registers)" and "–retiming (Register Retiming During Global Optimization)" options for use with –global_opt.

–gm (Guide Mode)

–gm {**exact** | **leverage**}

The –gm option specifies the form of guided mapping to be used**.**

In the EXACT mode the mapping in the guide file is followed exactly. In the LEVERAGE mode, the guide design is used as a starting point for mapping but, in cases where the guided design tools cannot find matches between net and block names in the input and guide designs, or your constraints rule out any matches, the logic is not guided.

For a description of guided mapping, see "Guided Mapping".

–gm incremental (Guide Mode incremental)

```
par -gf previous_run_NCD -gm incremental design.ncd
new_design.ncd design.pcf
```

The incremental guide mode uses EXACT guiding. It also changes the Partial Reconfiguration DRC checks.

Note: The Incremental Design flow is being deprecated and will not be available in future releases of Xilinx software. New in 8.2i are Partitions, which provide significant flexibility and functionality for design preservation. Information on Partitions can be found in the online help included in the 8.2i software and in the "TCL chapter" of this book. Incremental Design using the map and par -gm incremental option will still work in 8.2i, though it generates a warning that this flow is being removed.

–ignore_keep_hierarchy (Ignore KEEP_HIERARCHY Properties)

Map also supports the –ignore_keep_hierarchy option that ignores any "KEEP_HIERARCHY" properties on blocks.

–intstyle (Integration Style)

-intstyle {**ise** | **xflow** | **silent**}

The –intstyle option reduces screen output based on the integration style you are running. When using the –intstyle option, one of three modes must be specified: *ise*, *xflow*, or *silent*. The mode sets the way information is displayed in the following ways:

-intstyle ise

This mode indicates the program is being run as part of an integrated design environment.

-intstyle xflow

This mode indicates the program is being run as part of an integrated batch flow.

-intstyle silent

This mode limits screen output to warning and error messages only.

Note: The -intstyle option is automatically invoked when running in an integrated environment, such as Project Navigator or XFLOW.

–ir (Do Not Use RLOCs to Generate RPMs)

If you enter the –ir option, MAP uses RLOC constraints to group logic within CLBs, but does not use the constraints to generate RPMs (Relationally Placed Macros) controlling the relative placement of CLBs. Stated another way, the RLOCs are not used to control the relative placement of the CLBs with respect to each other.

For the Spartan architectures, the –ir option has an additional behavior; the RLOC constraint that cannot be met is ignored and the mapper will continue processing the design. A warning is generated for each RLOC that is ignored. The resulting mapped design is a valid design.

–ise (ISE Project File)

-ise *project_file*

The –ise option specifies an ISE project file, which can contain settings to capture and filter messages produced by the program during execution.

–k (Map to Input Functions)

The syntax for Spartan-II, Spartan-IIE, Virtex, and Virtex-E architectures follows:

–k {**4** | **5** | **6**}

The syntax for the Spartan 3/3E/3L, Virtex-4/-FX/-LX/-SX, Virtex-II, Virtex-II Pro/-X architectures follows:

–k {**4** |**5** |**6**| **7**| **8**}

You can specify the maximum size function that is covered. The default is 4. Covering to 5, 6, 7 or 8 input functions results in the use of F5MUX, F6MUX, and FXMUX.

By mapping input functions into single CLBs, the –k option may produce a mapping with fewer levels of logic, thus eliminating a number of CLB-to-CLB delays. However, using the –k option may prevent logic from being packed into CLBs in a way that minimizes CLB utilization.

For synthesis-based designs, specifying **–k 4** has no effect. This is because MAP combines smaller input functions into large functions such as F5MUX, F6MUX, F7MUX and F8MUX.

–l (No logic replication)

By default (without the –l option), MAP performs logic replication. Logic replication is an optimization method in which MAP operates on a single driver that is driving multiple loads and maps it as multiple components, each driving a single load (refer to the following figure). Logic replication results in a mapping that often makes it easier to meet your timing requirements, since some delays can be eliminated on critical nets. To turn off logic replication, you must specify the -l option.

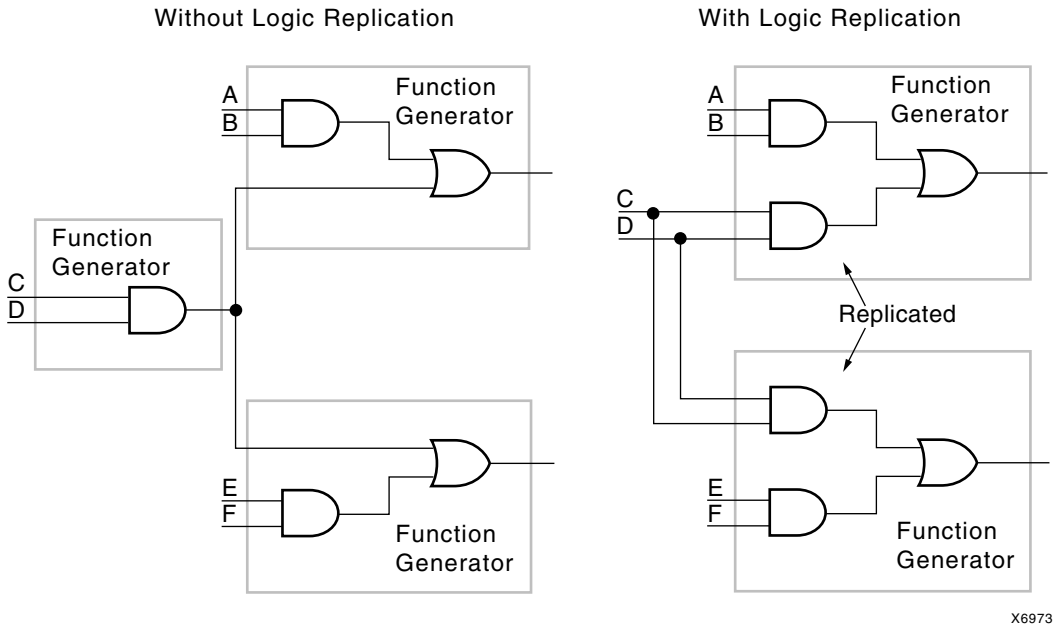


Figure 7-2: **Logic Replication (–l Option)**

–o (Output File Name)

–o *outfile*[**.ncd**]

Specifies the name of the output NCD file for the design. The .ncd extension is optional. The output file name and its location are determined in the following ways:

- If you do not specify an output file name with the –o option, the output file has the same name as the input file, with a .ncd extension. The file is placed in the input file's directory
- If you specify an output file name with no path specifier (for example, **cpu_dec.ncd** instead of **/home/designs/cpu_dec.ncd)**, the NCD file is placed in the current working directory.
- If you specify an output file name with a full path specifier (for example, **/home/designs/cpu_dec.ncd**), the output file is placed in the specified directory.

If the output file already exists, it is overwritten with the new NCD file. You do not receive a warning when the file is overwritten.

Note: However, signals connected to pads or to the outputs of BUFTs, flip-flops, latches, and RAMS are preserved for back-annotation.

–ol (Overall Effort Level)

–ol *[std | med | high]*

The –ol option is available when running timing-driven packing and placement with the –timing option. The –ol option sets the overall MAP effort level. The effort level specifies the level of effort MAP uses to pack the design.

Of the three *effort_level* values, use std for low effort level (fastest runtime at expense of QOR), use med for medium effort level (balance of runtime and QOR), use high for high effort level (best QOR with increased runtime).

The default effort level in MAP is high. Following is command line syntax for using the –ol option, set to std:

```
map –timing –ol std design.ncd output.ncd design.pcf
```

Note: The `–ol option is ignored if the –timing option is not set.`

–p (Part Number)

–p *part*

Specifies the Xilinx part number for the device. The syntax for the –p option is described in "–p (Part Number)" in Chapter 1.

If you do not specify a part number using the –p option, MAP selects the part specified in the input NGD file. If the information in the input NGD file does not specify a complete device and package, you must enter a device and package specification using the –p option. MAP supplies a default speed value, if necessary.

The architecture you specify with the –p option must match the architecture specified within the input NGD file. You may have chosen this architecture when you ran NGDBuild or during an earlier step in the design entry process (for example, you may have specified the architecture as an attribute within a schematic, or specified it as an option to a netlist reader). If the architecture does not match, you have to run NGDBuild again and specify the desired architecture.

You can only enter a part number or device name from a device library you have installed on your system. For example, if you have not installed the 4006E device library, you cannot create a design using the 4006E–PC84 part.

–pr (Pack Registers in I/O)

–pr {**i** | **o** | **b**}

By default (without the –pr option), MAP only places flip-flops or latches within an I/O cell if your design entry method specifies that these components are to be placed within I/O cells. For example, if you create a schematic using IFDX (Input D Flip-Flop) or OFDX (Output D Flip-Flop) design elements, the physical components corresponding to these design elements must be placed in I/O cells. The –pr option specifies that flip-flops or latches may be packed into input registers (**i** selection), output registers (**o** selection), or both (**b** selection) even if the components have not been specified in this way.

–r (No Register Ordering)

By default (without the –r option), MAP looks at the register bit names for similarities and tries to map register bits in an ordered manner (called *register ordering*). If you specify the -r option, register bit names are ignored when registers are mapped, and the bits are not mapped in any special order. For a description of register ordering, see "Register Ordering".

–register_duplication (Duplicate Registers)

The –register_duplication option is only available when running timing-driven packing and placement with the –timing option. The –register_duplication option duplicates registers to improve timing when running timing-driven packing. See "–timing (Timing-Driven Packing and Placement)."

–retiming (Register Retiming During Global Optimization)

-retiming *on|off*

When this option is *on*, registers are moved forward or backwards through the logic to balance out the delays in a timing path to increase the overall clock frequency. The overall number of registers may be altered due to the processing. This option is available for Virtex-4 designs. By default, this option is off.

Note: This option is available only when "–global_opt (Global Optimization)" is used.

–t (Start Placer Cost Table)

–t *placer_cost_table*

The –t option is only available when running timing-driven packing and placement with the –timing option. The –t option specifies the cost table at which the placer starts (placer cost tables are described in Chapter 9, "PAR"). The *placer_cost_table* range is 1–100. The default is 1.

–timing (Timing-Driven Packing and Placement)

Timing-driven packing and placement is recommended to improve design performance, timing, and packing for highly utilized designs. If the unrelated logic number (shown in the Design Summary section of the MAP report) is non-zero, then the –timing option is useful for packing more logic in the device. Timing-driven packing and placement is also recommended when there are local clocks present in the design.

Note: PAR issues a message to run timing-driven packing if it detects local clocks in the design. The –timing option is not supported on Virtex, Virtex-E, Spartan-II, and Spartan-IIE architectures.

The –timing option directs MAP to give priority to timing critical paths during packing, then places the design. Any user-generated timing constraints, contained in the UCF, drive the packing and placement operations. Use of the –timing option may result in longer runtime during the MAP process because designs are also placed; although, PAR runtime will be reduced since the placement phase is complete.

If Timing-driven packing and placement is selected in the absence of user timing constraints, the tools will automatically generate and dynamically adjust timing constraints for all internal clocks. This feature is referred to as "Performance Evaluation" mode. This mode allows the clock performance for all clocks in the design to be evaluated in one pass. The performance achieved by this mode is not necessarily the best possible performance each clock can achieve, instead it is a balance of performance between all clocks in the design.

The –ol option is used in conjunction with the –timing option to set the overall effort level that MAP uses to pack, and then place the design. See "–ol (Overall Effort Level)" for more information.

Note: The following options are specific to timing-driven packing and placement (–timing): –ol, –register_duplication, –t, and –xe. See individual option descriptions in this section for details.

–tx (Transform Buses)

–tx {**on** | **off** | **aggressive** | **limit**}

The –tx option specifies what type of bus transformation MAP performs. The four permitted settings are on, off, aggressive, and limit. The following example shows how the settings are used. In this example, the design has the following characteristics and is mapped to a Virtex device:

- Bus A has 4 BUFTs
- Bus B has 20 BUFTs
- Bus C has 30 BUFTs

MAP processes the design in one of the following ways, based on the setting used for the –tx option:

- The **on** setting performs partial transformation for a long chain that exceeds the device limit.
  - Bus A is transformed to LUTs (number of BUFTs is >1, ≤4)
  - Bus B is transformed to CY chain (number of BUFTs is >4, ≤48)
  - Bus C is *partially* transformed. (25 BUFTs + 1 dummy BUFT due to the maximum width of the XCV50 device + CY chain implementing the other 5 BUFTs)
- The **off** setting turns bus transformation off. This is the default setting.

- The **aggressive** setting transforms the entire bus.
  - Buses A, B have the same result as the *on* setting.
  - Bus C is implemented entirely by CY chain. (30 ≤ the default upper limit for carry chain transformation)
- The **limit** setting is the most conservative. It transforms only that portion of the number of CLB(s) or BUFT(s) per row in a device.

Note: The –tx option is not used for devices that do not have TBUFs, which include Virtex-4, Spartan-3, and Spartan-3E device families.

–u (Do Not Remove Unused Logic)

By default (without the –u option), MAP eliminates unused components and nets from the design before mapping. If –u is specified, MAP maps unused components and nets in the input design and includes them as part of the output design.

The –u option is helpful if you want to run a preliminary mapping on an unfinished design, possibly to see how many components the mapped design uses. By specifying –u, you are assured that all of the design's logic (even logic that is part of incomplete nets) is mapped.

–xe (Extra Effort Level)

–xe *effort_level*

The –xe option is available when running timing-driven packing and placement with the –timing option. The –xe option sets the extra effort level. The *effort_level* variable can be set to *n* (normal) or *c* (continue). Extra effort *c* allows you to direct MAP to continue packing. MAP continues to attempt to improve packing until little or no improvement can be made.

```
map –ol high –xe n design.ncd output.ncd design.pcf
```

MAP Process

MAP performs the following steps when mapping a design.

1. Selects the target Xilinx device, package, and speed. MAP selects a part in one of the following ways:
   - Uses the part specified on the MAP command line.
   - If a part is not specified on the command line, MAP selects the part specified in the input NGD file. If the information in the input NGD file does not specify a complete architecture, device, and package, MAP issues an error message and stops. If necessary, MAP supplies a default speed.
2. Reads the information in the input design file.
3. Performs a Logical DRC (Design Rule Check) on the input design. If any DRC errors are detected, the MAP run is aborted. If any DRC warnings are detected, the warnings are reported, but MAP continues to run. The Logical DRC (also called the NGD DRC) is described in Chapter 5, "Logical Design Rule Check".

   Note: Step 3 is skipped if the NGDBuild DRC was successful.

4. Removes unused logic. All unused components and nets are removed, unless the following conditions exist:
   - A Xilinx S (Save) constraint has been placed on a net during design entry. If an unused net has an S constraint, the net and all used logic connected to the net (as drivers or loads) is retained. All unused logic connected to the net is deleted. For a more complete description of the S constraint, see the *Constraints Guide*.
   - The –u option was specified on the MAP command line. If this option is specified, all unused logic is kept in the design.
5. Maps pads and their associated logic into IOBs.
6. Maps the logic into Xilinx components (IOBs, CLBs, etc.). If any Xilinx mapping control symbols appear in the design hierarchy of the input file (for example, FMAP symbols targeted to a Xilinx device), MAP uses the existing mapping of these components in preference to remapping them. The mapping is influenced by various constraints; these constraints are described in the *Constraints Guide*.
7. Update the information received from the input NGD file and write this updated information into an NGM file. This NGM file contains both logical information about the design and physical information about how the design was mapped. The NGM file is used only for back-annotation. On Virtex/-E/-II devices, guided mapping uses the NGM file. For more information, see "Guided Mapping".
8. Create a physical constraints (PCF) file. This is a text file that contains any constraints specified during design entry. If no constraints were specified during design entry, an empty file is created so that you can enter constraints directly into the file using a text editor or indirectly through the FPGA Editor.

   MAP either creates a PCF file if none exists or rewrites an existing file by overwriting the schematic-generated section of the file (between the statements SCHEMATIC START and SCHEMATIC END). For an existing constraints file, MAP also checks the user-generated section and may either comment out constraints with errors or halt the program. If no errors are found in the user-generated section, the section remains the same.

   Note: For Virtex/-E/-II/-II PRO designs, you must use a MAP generated PCF file. The timing tools perform skew checking only with a MAP-generated PCF file.
9. Run a physical Design Rule Check (DRC) on the mapped design. If DRC errors are found, MAP does not write an NCD file.
10. Create an NCD file, which represents the physical design. The NCD file describes the design in terms of Xilinx components—CLBs, IOBs, etc.
11. Write a MAP report (MRP) file, which lists any errors or warnings found in the design, details how the design was mapped, and supplies statistics about component usage in the mapped design.

Register Ordering

When you run MAP, the default setting performs register ordering. If you specify the –r option, MAP does not perform register ordering and maps the register bits as if they were unrelated.

When you map a design containing registers, the MAP software can optimize the way the registers are grouped into CLBs (slices for Virtex/-E/-II/-II PRO or Spartan-II/-IIE — there are two slices per CLB). This optimized mapping is called *register ordering*.

A CLB (Virtex/-E/-II/-II PRO or Spartan-II/IIE slice) has two flip-flops, so two register bits can be mapped into one CLB. For PAR (Place And Route) to place a register in the most effective way, you want as many pairs of contiguous bits as possible to be mapped together into the same CLBs (for example, bit 0 and bit 1 together in one CLB, bit 2 and bit 3 in another).

MAP pairs register bits (performing *register ordering*) if it recognizes that a series of flip-flops comprise a register. When you create your design, you can name register bits so they are mapped using register ordering.

Note: MAP *does not* perform register ordering on any flip-flops which have BLKNM, LOC, or RLOC properties attached to them. The BLKNM, LOC, and RLOC properties define how blocks are to be mapped, and these properties override register ordering.

To be recognized as a candidate for register ordering, the flip-flops must have the following characteristics:

- The flip-flops must share a common clock signal and common control signals (for example, Reset and Clock Enable).
- The flip-flop output signals must all be named according to this convention.
- Output signal names must begin with a common root containing at least one alphabetic character.

  The names must end with numeric characters or with numeric characters surrounded by parentheses "( )", angle brackets "< >", or square brackets "[ ]".

  For example, acceptable output signal names for register ordering are as follows:

data1	**addr(04)**	**bus<1>**
data2	**addr(08)**	**bus<2>**
data3	**addr(12)**	**bus<3>**
data4	**addr(16)**	**bus<4>**

If a series of flip-flops is recognized as a candidate for register ordering, they are paired in CLBs in sequential numerical order. For example, in the first set of names shown above, data1 and data2, are paired in one CLB, while data3 and data4 are paired in another.

In the example below, no register ordering is performed, since the root names for the signals are not identical

data01

addr02

atod03

dtoa04

When it finds a signal with this type of name, MAP ignores the underscore and the numeric characters when it considers the signal for register ordering. For example, if signals are named data00_1 and data01_2, MAP considers them as data00 and data01 for purposes of register ordering. These two signals *are* mapped to the same CLB.

MAP does not change signal names when it checks for underscores—it only ignores the underscore and the number when it checks to see if the signal is a candidate for register ordering.

Because of the way signals are checked, make sure you don't use an underscore as your bus delimiter. If you name a bus signal data0_01 and a non-bus signal data1, MAP sees them as data0 and data1 and register orders them even though you do not want them register ordered.

Guided Mapping

In guided mapping, an existing NCD is used to guide the current MAP run. The guide file may be from any stage of implementation: unplaced or placed, unrouted or routed. Xilinx recommends that you generate your NCD file using the current release of the software; however, MAP does support guided mapping using NCD files from the previous releases.

Note: When using guided mapping with the –timing option, Xilinx recommends using a placed NCD as the guide file. A placed NCD is produced by running MAP –timing or PAR.

The following figure shows the guided mapping flow:

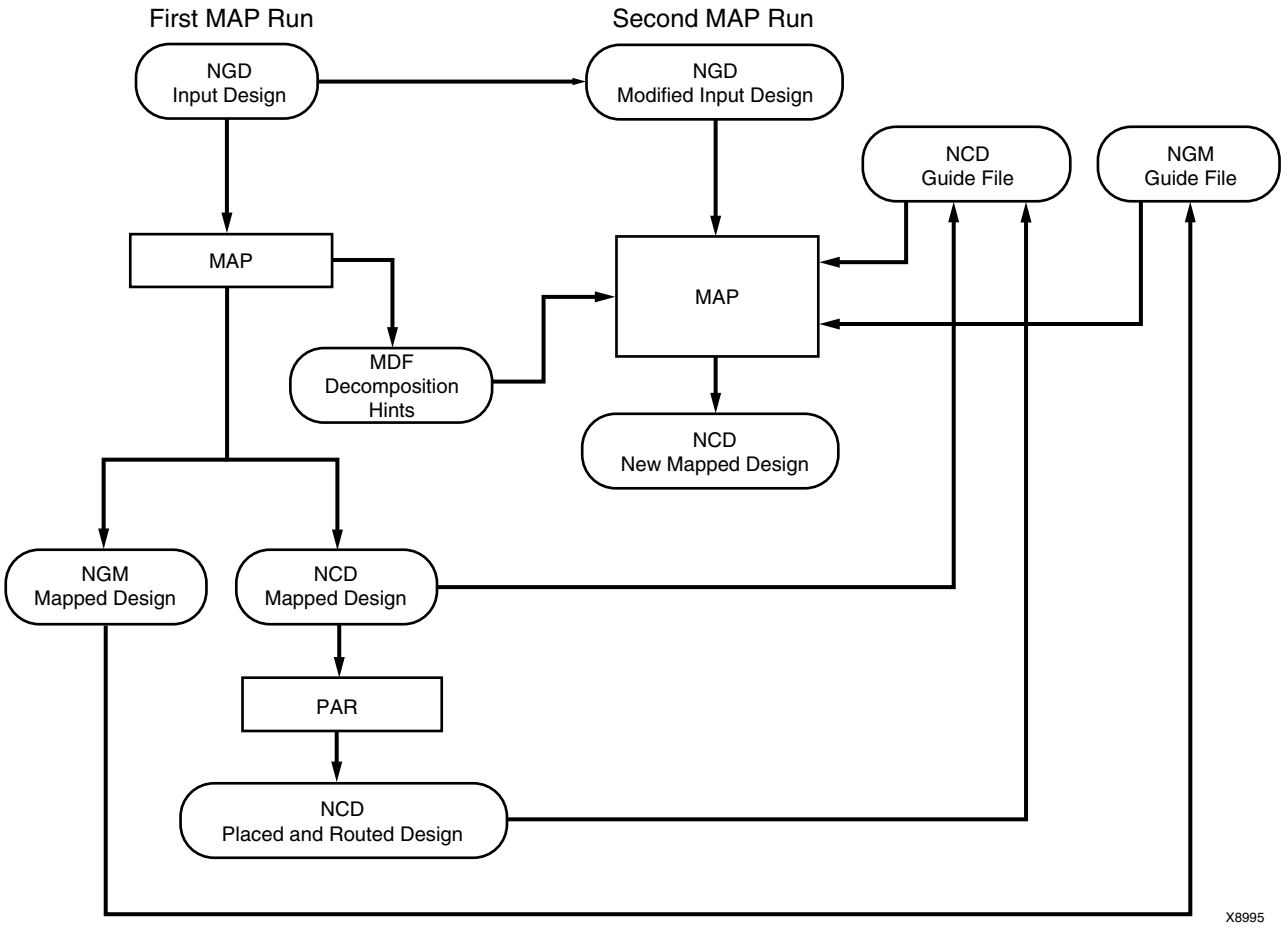


Figure 7-3: **Guided Mapping**

In the EXACT mode the mapping in the guide file is followed exactly. Any logic in the input NGD file that matches logic mapped into the physical components of the NCD guide file is implemented exactly as in the guide file. Mapping (including signal to pin assignments), placement and routing are all identical. Logic that is not matched to any guide component is mapped by a subsequent mapping step.

If there is a match in EXACT mode, but your constraints would conflict with the mapping in the guide file component, an error is posted. If an error is posted, you can do one of the following:

- Modify the constraints to eliminate conflicts
- Change to the LEVERAGE guide mode (which is less restrictive)
- Modify the logical design changes to avoid conflicts
- Stop using guided design

In the LEVERAGE mode, the guide design is used as a starting point in order to speed up the design process. However, in cases where the guided design tools cannot find matches or your constraints rule out any matches, the logic is not guided. Whenever the guide design conflicts with the your mapping, placement or routing constraints, the guide is ignored and your constraints are followed.

Because the LEVERAGE mode only uses the guide design as a starting point for mapping, MAP may alter the mapping to improve the speed or density of the implementation (for example, MAP may collapse additional gates into a guided CLB).

Note: Support for the leverage guide flow (–gm incremental), without a timing-driven map run of your design, specified with the map –timing option, will not be supported in future releases of Xilinx software.

For Spartan and Virtex/-E/-II/-II PRO devices, MAP uses the NGM and the NCD files as guides. You do not need to specify the NGM file on the command line. MAP infers the appropriate NGM file from the specified NCD file. If MAP does not find an NGM file in the same directory as the NCD, it generates a warning. In this case, MAP uses only the NCD file as the guide file.

Note: Guided mapping is not recommended for most HDL designs. Guided mapping depends on signal and component names, and HDL designs often have a low *match rate* when guided. The netlist produced after re-synthesizing HDL modules usually contains signal and instance names that are significantly different from netlists created by earlier synthesis runs. This occurs even if the source level HDL code contains only a few changes.

Simulating Map Results

When simulating with NGM files, you are not simulating a mapped result, you are simulating the logical circuit description. When simulating with NCD files, you are simulating the physical circuit description.

MAP may generate an error that is not detected in the back-annotated simulation netlist. For example, after running MAP, you can run the following command to generate the back-annotated simulation netlist:

```
netgen mapped.ncd mapped.ngm -o mapped.nga
```

This command creates a back-annotated simulation netlist using the logical-to-physical cross-reference file named mapped.ngm. This cross-reference file contains information about the logical design netlist, and the back-annotated simulation netlist (mapped.nga) is actually a back-annotated version of the logical design. However, if MAP makes a physical error, for example, implements an Active Low function for an Active High function, this error will not be detected in the mapped.nga file and will not appear in the simulation netlist.

For example, consider the following logical circuit generated by NGDBuild from a design file, shown in the following figure.

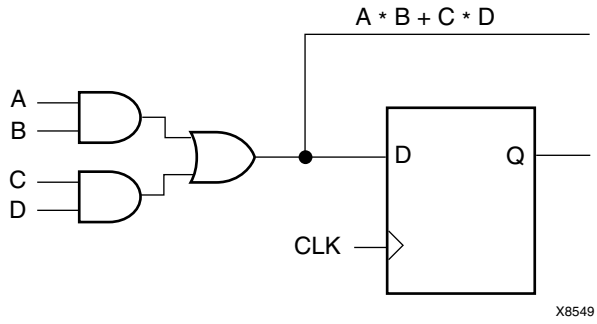


Figure 7-4: **Logical Circuit Representation**

Observe the Boolean output from the combinatorial logic. Suppose that after running MAP for the preceding circuit, you obtain the following result.

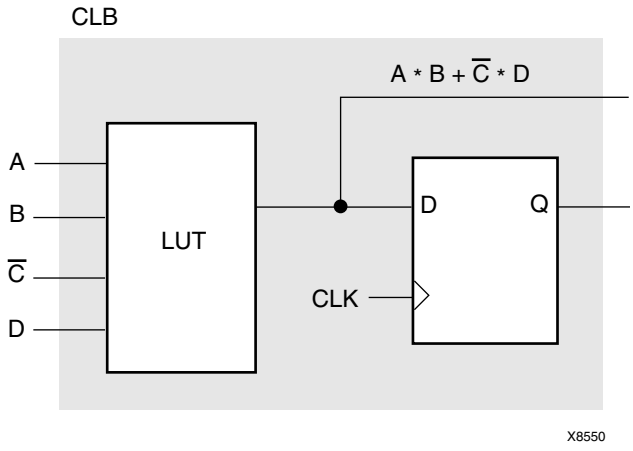


Figure 7-5: **CLB Configuration**

Observe that MAP has generated an active low (C) instead of an active high (C). Consequently, the Boolean output for the combinatorial logic is incorrect. When you run NetGen using the mapped.ngm file, you cannot detect the logical error because the delays are back-annotated to the correct logical design, and not to the physical design.

One way to detect the error is by running the NetGen command without using the mapped.ngm cross-reference file.

```
netgen mapped.ncd -o mapped.nga
```

As a result, physical simulations using the mapped.nga file should detect a physical error. However, the type of error is not always easily recognizable. To pinpoint the error, use the FPGA Editor or call Xilinx Customer Support. In some cases, a reported error may not really exist, and the CLB configuration is actually correct. You can use the FPGA Editor to determine if the CLB is correctly modelled.

Finally, if both the logical and physical simulations do not discover existing errors, you may need to use more test vectors in the simulations.

MAP Report (MRP) File

The MAP report (MRP) file is an ASCII text file that contains information about the MAP run. The report information varies based on the device and whether you use the –detail option (see the "–detail (Write Out Detailed MAP Report)" section).

An abbreviated MRP file is shown below—most report files are considerably larger than the one shown. The file is divided into a number of sections, and sections appear even if they are empty. The sections of the MRP file are as follows:

- Design Information—Shows your MAP command line, the device to which the design has been mapped, and when the mapping was performed.
- Design Summary—Summarizes the mapper run, showing the number of errors and warnings, and how many of the resources in the target device are used by the mapped design.
- Table of Contents—Lists the remaining sections of the MAP report.
- Errors—Shows any errors generated as a result of the following:
  - Errors associated with the logical DRC tests performed at the beginning of the mapper run. These errors do not depend on the device to which you are mapping.
  - Errors the mapper discovers (for example, a pad is not connected to any logic, or a bidirectional pad is placed in the design but signals only pass in one direction through the pad). These errors may depend on the device to which you are mapping.
  - Errors associated with the physical DRC run on the mapped design.
- Warnings—Shows any warnings generated as a result of the following:
  - Warnings associated with the logical DRC tests performed at the beginning of the mapper run. These warnings do not depend on the device to which you are mapping.
  - Warnings the mapper discovers. These warnings may depend on the device to which you are mapping.
  - Warnings associated with the physical DRC run on the mapped design.
- Informational—Shows messages that usually do not require user intervention to prevent a problem later in the flow. These messages contain information that may be valuable later if problems do occur.
- Removed Logic Summary—Summarizes the number of blocks and signals removed from the design. The section reports on these kinds of removed logic.

- Blocks trimmed—A trimmed block is removed because it is along a path that has no driver or no load. Trimming is recursive. For example, if Block A becomes unnecessary because logic to which it is connected has been trimmed, then Block A is also trimmed.
  - Blocks removed—A block is removed because it can be eliminated without changing the operation of the design. Removal is recursive. For example, if Block A becomes unnecessary because logic to which it is connected has been removed, then Block A is also removed.
  - Blocks optimized—An optimized block is removed because its output remains constant regardless of the state of the inputs (for example, an AND gate with one input tied to ground). Logic generating an input to this optimized block (and to no other blocks) is also removed, and appears in this section.
  - Signals removed—Signals are removed if they are attached only to removed blocks.
  - Signals merged—Signals are merged when a component separating them is removed.
- Removed Logic—Describes in detail all logic (design components and nets) removed from the input NGD file when the design was mapped. Generally, logic is removed for the following reasons:
  - The design uses only part of the logic in a library macro.
  - The design has been mapped even though it is not yet complete.
  - The mapper has optimized the design logic.
  - Unused logic has been created in error during schematic entry.

  This section also indicates which nets were merged (for example, two nets were combined when a component separating them was removed).

  In this section, if the removal of a signal or symbol results in the subsequent removal of an additional signal or symbol, the line describing the subsequent removal is indented. This indentation is repeated as a chain of related logic is removed. To quickly locate the cause for the removal of a chain of logic, look above the entry in which you are interested and locate the top-level line, which is not indented.
- IOB Properties—Lists each IOB to which the user has supplied constraints along with the applicable constraints.
- RPMs—Indicates each RPM (Relationally Placed Macro) used in the design, and the number of device components used to implement the RPM.
- Guide Report—If you have mapped using a guide file, shows the guide mode used (EXACT or LEVERAGE) and the percentage of objects that were successfully guided.
- Area Group Summary—The mapper summarizes results for each area group. MAP uses area groups to specify a group of logical blocks that are packed into separate physical areas.
- Modular Design Summary—After the Modular Design Active Module Implementation Phase, this section lists the logic that was added to the design to successfully implement the active module. After the Final Assembly Phase, this section states whether the logic was assembled successfully.
- Timing Report—This section, produced with the –timing option, shows information on timing constraints considered during the MAP run.

- Configuration String Information—This section, produced with the -detail option, shows configuration strings and programming properties for *special* components like DCMs, BRAMS, GTs and similar components. Configuration strings for slices and IOBs marked “SECURE” are not shown.
- Additional Device Resource Counts—This section contains raw design statistics for Xilinx analysis purposes.

Note: The MAP Report is formatted for viewing in a monospace (non-proportional) font. If the text editor you use for viewing the report uses a proportional font, the columns in the report do not line up correctly.

```
Release 8.1i Map
Xilinx Mapping Report File for Design 'stopwatch'

Design Information
------------------
Command Line   : C:/xilinx/bin/nt/map.exe -ise
c:\xilinx\projects\watchver\watchver.ise
-intstyle ise -p xc2v40-fg256-5 -cm area -pr b -k 4 -c 100 -tx off -o
stopwatch_map.ncd stopwatch.ngd stopwatch.pcf
Target Device  : xc2v40
Target Package : fg256
Target Speed   : -5
Mapper Version : virtex2 -- $Revision: 1.26.6.3 $
Mapped Date    : Mon Nov 01 18:11:26 2005

Design Summary
--------------
Number of errors:      0
Number of warnings:    3
Logic Utilization:
  Number of Slice Flip Flops:          17 out of     512    3%
  Number of 4 input LUTs:              54 out of     512   10%
Logic Distribution:
  Number of occupied Slices:           29 out of     256   11%
  Number of Slices containing only related logic:     29 out of 29  100%
  Number of Slices containing unrelated logic:         0 out of 29    0%
Total Number 4 input LUTs:             54 out of     512   10%

  Number of bonded IOBs:               27 out of      88   30%
  Number of GCLKs:                      1 out of      16    6%
  Number of DCMs:                       1 out of       4   25%

   Number of RPM macros:            1
Total equivalent gate count for design:  7,487
Additional JTAG gate count for IOBs:  1,296
Peak Memory Usage:  98 MB
```

```
Table of Contents
-----------------
Section 1 - Errors
Section 2 - Warnings
Section 3 - Informational
Section 4 - Removed Logic Summary
Section 5 - Removed Logic
Section 6 - IOB Properties
Section 7 - RPMs
Section 8 - Guide Report
Section 9 - Area Group Summary
Section 10 - Modular Design Summary
Section 11 - Timing Report
Section 12 - Configuration String Information
Section 13 - Additional Device Resource Counts

Section 1 - Errors
------------------

Section 2 - Warnings
--------------------
WARNING:LIT - Logical network Inst_dcm1_CLKIN_IBUFG_OUT has no load.
WARNING:LIT - The above warning message base_net_load_rule is repeated
1 more time for the following:
   N0
   To see the details of these warning messages, please use the -detail
   switch.

Section 3 - Informational
-------------------------
INFO:MapLib:562 - No environment variables are currently set.
INFO:LIT:244 - All of the single ended outputs in this design are using
slew rate limited output drivers. The delay on speed critical single
ended outputs can be dramatically reduced by designating them as fast
outputs in the schematic.

Section 4 - Removed Logic Summary
---------------------------------
   1 block(s) removed
   2 block(s) optimized away

Section 5 - Removed Logic
-------------------------
Unused block "xcounter/VCC" (ONE) removed.

Optimized Block(s):
TYPE            BLOCK
GND             XST_GND
GND             xcounter/GND

To enable printing of redundant blocks removed and signals merged, set
the
detailed map report option and rerun map.
```

Section 6 - IOB Properties

IOB Name	Type	Direction	I/O Standard	Drive Strength	Slew Rate	Reg(s)	Register	IOB Delay
CLK	IOB	Input	LVTTL					
ONESOUT<0>	IOB	Output	LVTTL	12	SLOW			
ONESOUT<1>	IOB	Output	LVTTL	12	SLOW			
ONESOUT<2>	IOB	Output	LVTTL	12	SLOW			
ONESOUT<3>	IOB	Output	LVTTL	12	SLOW			
ONESOUT<4>	IOB	Output	LVTTL	12	SLOW			
ONESOUT<5>	IOB	Output	LVTTL	12	SLOW			
ONESOUT<6>	IOB	Output	LVTTL	12	SLOW			
RESET	IOB	Input	LVTTL					
STRTSTOP	IOB	Input	LVTTL					
TENSOUT<0>	IOB	Output	LVTTL	12	SLOW			
TENSOUT<1>	IOB	Output	LVTTL	12	SLOW			
TENSOUT<2>	IOB	Output	LVTTL	12	SLOW			
TENSOUT<3>	IOB	Output	LVTTL	12	SLOW			
TENSOUT<4>	IOB	Output	LVTTL	12	SLOW			
TENSOUT<5>	IOB	Output	LVTTL	12	SLOW			
TENSOUT<6>	IOB	Output	LVTTL	12	SLOW			
TENTHSOUT<0>	IOB	Output	LVTTL	12	SLOW			
TENTHSOUT<1>	IOB	Output	LVTTL	12	SLOW			
TENTHSOUT<2>	IOB	Output	LVTTL	12	SLOW			
TENTHSOUT<3>	IOB	Output	LVTTL	12	SLOW			
TENTHSOUT<4>	IOB	Output	LVTTL	12	SLOW			
TENTHSOUT<5>	IOB	Output	LVTTL	12	SLOW			
TENTHSOUT<6>	IOB	Output	LVTTL	12	SLOW			
TENTHSOUT<7>	IOB	Output	LVTTL	12	SLOW			
TENTHSOUT<8>	IOB	Output	LVTTL	12	SLOW			
TENTHSOUT<9>	IOB	Output	LVTTL	12	SLOW			

```
Section 7 - RPMs
----------------
xcounter/hset

Section 8 - Guide Report
------------------------
Guide not run on this design.

Section 9 - Area Group Summary
------------------------------
No area groups were found in this design.

Section 10 - Modular Design Summary
-----------------------------------
Modular Design not used for this design.

Section 11 - Timing Report
--------------------------
This design was not run using timing mode.

Section 12 - Configuration String Details
-------------------------
Use the "-detail" map option to print out Configuration Strings

Section 13 - Additional Device Resource Counts
----------------------------------------------
Number of JTAG Gates for IOBs = 27
Number of Equivalent Gates for Design = 7,487
Number of RPM Macros = 1
Number of Hard Macros = 0
CAPTUREs = 0
BSCANs = 0
STARTUPs = 0
PCILOGICs = 0
DCMs = 1
GCLKs = 1
ICAPs = 0
18X18 Multipliers = 0
Block RAMs = 0
TBUFs = 0
Total Registers (Flops & Latches in Slices & IOBs) not driven by LUTs=3
IOB Dual-Rate Flops not driven by LUTs = 0
IOB Dual-Rate Flops = 0
IOB Slave Pads = 0
IOB Master Pads = 0
IOB Latches not driven by LUTs = 0
IOB Latches = 0
IOB Flip Flops not driven by LUTs = 0
IOB Flip Flops = 0
Unbonded IOBs = 0
Bonded IOBs = 27
Total Shift Registers = 0
Static Shift Registers = 0
Dynamic Shift Registers = 0
16x1 ROMs = 0
16x1 RAMs = 0
32x1 RAMs = 0
Dual Port RAMs = 0
MUXFs = 1
```

```
MULT_ANDs = 0
4 input LUTs used as Route-Thrus = 0
4 input LUTs = 54
Slice Latches not driven by LUTs = 0
Slice Latches = 0
Slice Flip Flops not driven by LUTs = 3
Slice Flip Flops = 17
Slices = 29
Number of LUT signals with 4 loads = 4
Number of LUT signals with 3 loads = 0
Number of LUT signals with 2 loads = 4
Number of LUT signals with 1 load = 44
NGM Average fanout of LUT = 1.52
NGM Maximum fanout of LUT = 9
NGM Average fanin for LUT = 3.4444
Number of LUT symbols = 54
```

Halting MAP

To halt MAP, enter Ctrl+C (on a workstation) or Ctrl+Break (on a PC). On a workstation, make sure that when you enter Ctrl+C the active window is the window from which you invoked the mapper. The operation in progress is halted. Some files may be left when the mapper is halted (for example, a MAP report file or a physical constraints file), but these files may be discarded since they represent an incomplete operation.

Chapter 8

Physical Design Rule Check

This program is compatible with the following families:

- Virtex™, Virtex™-E
- Virtex™-II
- Virtex™-II Pro, Virtex™-II Pro X
- Virtex™-4
- Virtex™-5 LX
- Spartan™-II, Spartan™-IIE
- Spartan™-3, Spartan™-3E, Spartan™-3L

The chapter describes the physical Design Rule Check program. This chapter contains the following sections:

DRC Overview

The physical Design Rule Check, also known as DRC, comprises a series of tests to discover physical errors and some logic errors in the design. The physical DRC is run as follows:

- MAP automatically runs physical DRC after it has mapped the design.
- PAR (Place and Route) automatically runs physical DRC on nets when it routes the design.
- BitGen, which creates a a BIT file for programming the device, automatically runs physical DRC.
- You can run physical DRC from within the FPGA Editor tool. The DRC also runs automatically after certain FPGA Editor operations (for example, when you edit a logic cell or when you manually route a net). For a description of how the DRC works within the FPGA Editor, see the online help provided with the FPGA Editor GUI tool.
- You can run physical DRC from the UNIX or DOS command line.

DRC Syntax

The following command runs physical DRC:

drc [*options*] *file_name*.**ncd**

options can be any number of the DRC options listed in "DRC Options". They do not need to be listed in any particular order. Separate multiple options with spaces.

file_name is the name of the NCD file on which DRC is to be run.

DRC Input File

The input to DRC is an NCD file. The NCD file is a mapped, physical description of your design.

DRC Output File

The output of DRC is a TDR file. The TDR file is an ASCII formatted DRC report. The contents of this file are determined by the command line options you specify with the DRC command.

DRC Options

This section describes the DRC command line options.

–e (Error Report)

The –e option produces a report containing details about errors only. No details are given about warnings.

–o (Output file)

–o *outfile_name*.**tdr**

The –o option overrides the default output report file *file_name*.tdr with *outfile_name*.tdr.

–s (Summary Report)

The –s option produces a summary report only. The report lists the number of errors and warnings found but does not supply any details about them.

–v (Verbose Report)

The –v option reports all warnings and errors. This is the default option for DRC.

–z (Report Incomplete Programming)

The –z option reports incomplete programming as errors. Certain DRC violations are considered errors when the DRC runs as part of the BitGen command but are considered warnings at all other times the DRC runs. These violations usually indicate the design is incompletely programmed (for example, a logic cell has been only partially programmed or a signal has no driver). The violations create errors if you try to program the device, so they are reported as errors when BitGen creates a BIT file for device programming. If you run DRC from the command line without the –z option, these violations are reported as warnings only. With the –z option, these violations are reported as errors.

DRC Checks

Physical DRC performs the following types of checks:

- Net check

  This check examines one or more routed or unrouted signals and reports any problems with pin counts, 3-state buffer inconsistencies, floating segments, antennae, and partial routes.

- Block check

  This check examines one or more placed or unplaced components and reports any problems with logic, physical pin connections, or programming.

- Chip check

  This check examines a special class of checks for signals, components, or both at the chip level, such as placement rules with respect to one side of the device.

- All checks

  This check performs net, block, and chip checks.

When you run DRC from the command line, it automatically performs net, block, and chip checks.

In the FPGA Editor, you can run the net check on selected objects or on all of the signals in the design. Similarly, the block check can be performed on selected components or on all of the design's components. When you check all components in the design, the block check performs extra tests on the design as a whole (for example, 3-state buffers sharing long lines and oscillator circuitry configured correctly) in addition to checking the individual components. In the FPGA Editor, you can run the net check and block check separately or together.

DRC Errors and Warnings

A DRC error indicates a condition in which the routing or component logic does not operate correctly (for example, a net without a driver or a logic block that is incorrectly programmed). A DRC warning indicates a condition where the routing or logic is incomplete (for example, a net is not fully routed or a logic block has been programmed to process a signal but there is no signal on the appropriate logic block pin).

Certain messages may appear as either warnings or errors, depending on the application and signal connections. For example, in a net check, a pull-up not used on a signal connected to a decoder generates an error message. A pull-up not used on a signal connected to a 3-state buffer only generates a warning.

Incomplete programing (for example, a signal without a driver or a partially programmed logic cell) is reported as an error when the DRC runs as part of the BitGen command, but is reported as a warning when the DRC runs as part of any other program. The –z option to the DRC command reports incomplete programming as an error instead of a warning. For a description of the –z option, see "–z (Report Incomplete Programming)".

Chapter 9

PAR

The Place and Route (PAR) program is compatible with the following families:

- Virtex™, Virtex™-E
- Virtex™-II
- Virtex™-II Pro, Virtex™-II Pro X
- Virtex™-4
- Virtex™-5 LX
- Spartan™-II, Spartan™-IIE
- Spartan™-3, Spartan™-3E, Spartan™-3L

The chapter contains the following sections:

Place and Route Overview

After you create a Native Circuit Description (NCD) file with the MAP program, you can place and route that design file using PAR. PAR accepts a mapped NCD file as input, places and routes the design, and outputs an NCD file to be used by the bitstream generator (BitGen). See Chapter 14, "BitGen".

The NCD file output by PAR can also be used as a guide file for additional runs of PAR that may be done after making minor changes to your design. See the "Guided PAR" section of this chapter for more information on using guide files.

PAR places and routes a design based on the following considerations:

- Timing-driven—The Xilinx timing analysis software enables PAR to place and route a design based upon timing constraints. See the "Timing-driven PAR" section of this chapter for more information.
- Non Timing-driven (cost-based)—Placement and routing are performed using various cost tables that assign weighted values to relevant factors such as constraints, length of connection, and available routing resources. Non timing-driven placement and routing is used if no timing constraints are present.

The design flow through PAR is shown in the following figure. This figure shows a PAR run that produces a single output design file (NCD)

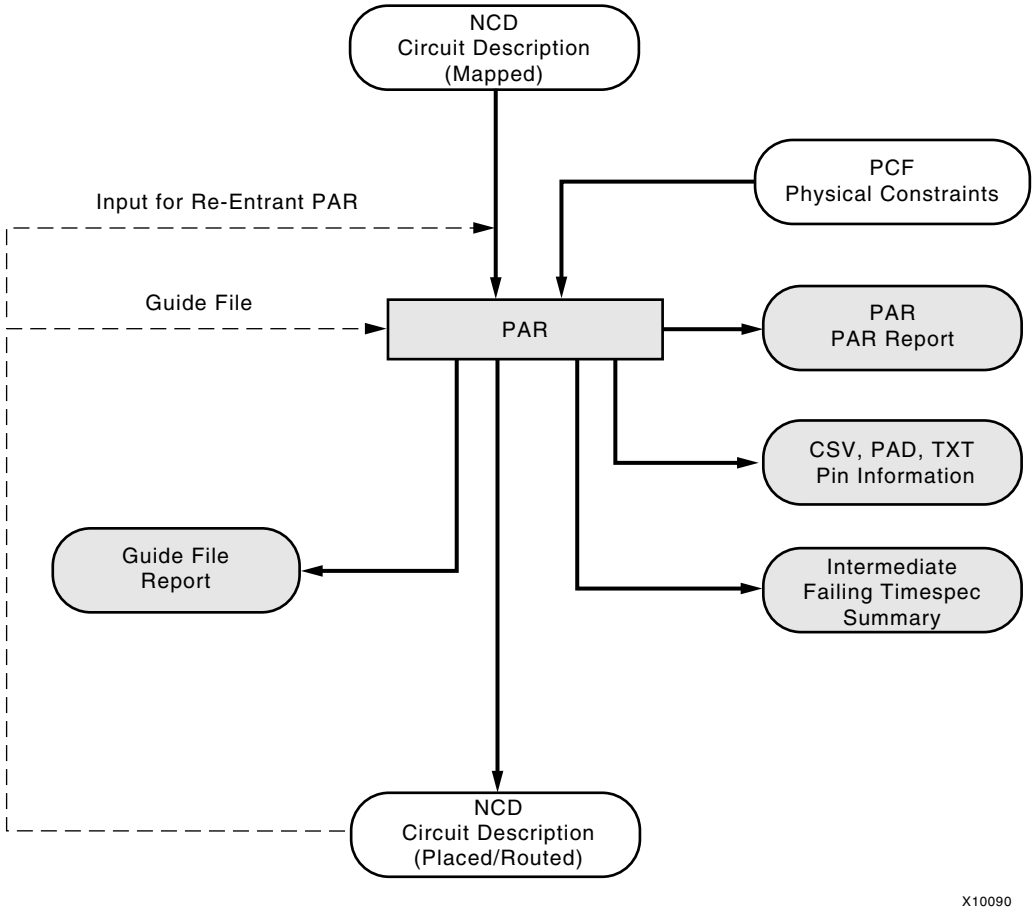


Figure 9-1: **PAR Flow**

PAR Process

This section provides information on how placing and routing are performed by PAR, as well as information on timing-driven PAR and automatic timespecing.

Placing

The PAR placer executes multiple phases of the placer. PAR writes the NCD after all the placer phases are complete.

During placement, PAR places components into sites based on factors such as constraints specified in the PCF file, the length of connections, and the available routing resources.

Routing

After placing the design, PAR executes multiple phases of the router. The router performs a converging procedure for a solution that routes the design to completion and meets timing constraints. Once the design is fully routed, PAR writes an NCD file, which can be analyzed against timing.

PAR writes a new NCD as the routing improves throughout the router phases.

Note: Timing-driven place and timing-driven routing are automatically invoked if PAR finds timing constraints in the physical constraints file. See the following section for details.

Timing-driven PAR

Timing-driven PAR is based on the Xilinx timing analysis software, an integrated static timing analysis tool that does not depend on input stimulus to the circuit. Placement and routing are executed according to timing constraints that you specify in the beginning of the design process. The timing analysis software interacts with PAR to ensure that the timing constraints imposed on your design are met.

To use timing-driven PAR, you can specify timing constraints using any of the following ways:

- Enter the timing constraints as properties in a schematic capture or HDL design entry program. In most cases, an NCF will be automatically generated by the synthesis tool.
- Write your timing constraints into a User Constraints File (UCF). This file is processed by NGDBuild when the logical design database is generated.

  To avoid manually entering timing constraints in a UCF, use the Xilinx Constraints Editor, a tool that greatly simplifies creating constraints. For a detailed description of how to use the Constraints Editor, see the Constraints Editor online help included with the software.
- Enter the timing constraints in the physical constraints file (PCF), a file that is generated by MAP. The PCF file contains any timing constraints specified using the two previously described methods and any additional constraints you enter in the file.

If no timing constraints are found for the design or the Project Navigator "Use Timing Constraints" option is unchecked (-x option), timing constraints are automatically generated for all internal clocks. These constraints will be adjusted to get better performance as PAR runs. The level of performance achieved is in direct relation to the setting of the PAR effort level. Effort level STD will have the fastest run time and the lowest performance, effort level HIGH will have the best performance and the longest run time, effort level MED will have run time and performance between STD and HIGH.

Timing-driven placement and timing-driven routing are automatically invoked if PAR finds timing constraints in the physical constraints file. The physical constraints file serves as input to the timing analysis software. The timing constraints supported by the Xilinx Development System are described in the *Constraints Guide*.

Note: Depending upon the types of timing constraints specified and the values assigned to the constraints, PAR run time may be increased.

When PAR is complete, you can review the output PAR Report for a timing summary or verify that the design's timing characteristics (relative to the physical constraints file) have been met by running TRACE (Timing Reporter and Circuit Evaluator) or Timing Analyzer, Xilinx's timing verification and reporting utilities. TRACE, which is described in detail in Chapter 12, "TRACE", issues a report showing any timing warnings and errors and other information relevant to the design.

Command Line Examples

Following are a few examples of PAR command line syntax and a description of what each does.

Example 1:

The following command places and routes the design in the file *input*.ncd and writes the placed and routed design to *output*.ncd.

par input.ncd output.ncd

Note: PAR will automatically detect and include a physical constraints file (PCF) that has the same root name as the input NCD file.

Example 2:

The following command skips the placement phase and preserves all routing information without locking it (re-entrant routing). Then it runs in conformance to timing constraints found in the pref.pcf file. If the design is fully routed and your timing constraints are not met, then the router attempts to reroute until timing goals are achieved or until it determines it is not achievable.

par –k previous.ncd reentrant.ncd pref.pcf

Example 3:

The following command runs 20 placements and routings using different cost tables all at overall effort level med. The mapping of the overall level (–ol) to placer effort level (–pl) and router effort level (–rl) depends on the device to which the design was mapped, and placer level and router level do not necessarily have the same value. The iterations begin at cost table entry 5. Only the best 3 output design files are saved. The output design files (in NCD format) are placed into a directory called results.dir.

```
par –n 20 –ol med –t 5 –s 3 input.ncd results.dir
```

Example 4:

The following command runs PAR (using the Turns Engine) on all nodes listed in the *allnodes* file. It runs 10 place and route passes at placer effort level med and router effort level std on the mydesign.ncd file.

par –m allnodes –pl med –rl std –n 10 mydesign.ncd output.dir

Note: This command is not supported on Windows operating systems.

Guided PAR

When PAR runs using a guide design as input, PAR first places and routes any components and signals that fulfill the matching criteria from the guide file.

Optionally, PAR reads a previously placed and routed NCD file as a guide file to help in placing and routing the input design. This is useful if minor incremental changes have been made to create a new design. To increase productivity, you can use your last design iteration as a guide design for the next design iteration, as shown in the following figure:

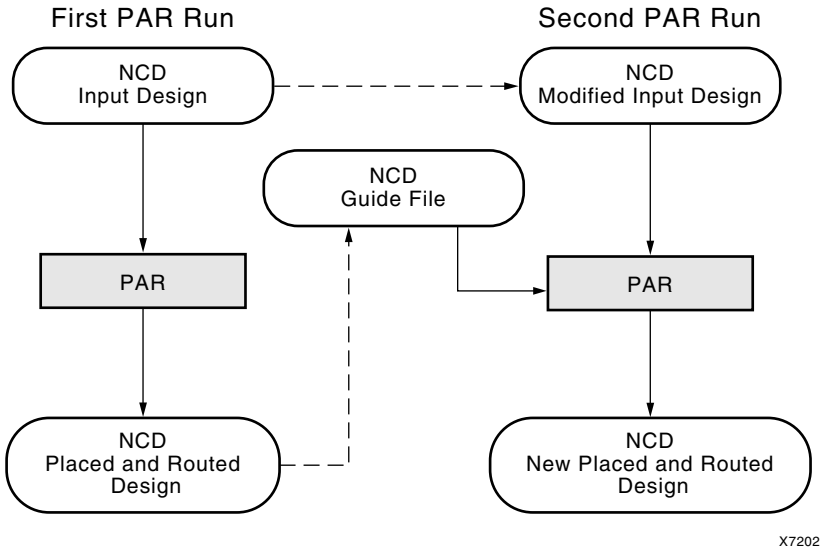


Figure 9-2: **Guided PAR for Design**

Two command line options control guided PAR. The –gf option specifies the NCD guide file, and the –gm option determines whether exact or leverage or incremental mode is used to guide PAR.

The guide design is used as follows:

- If a component in the new design is constrained to the same location as a component placed in the guide file, then this component is defined as matching.
- If a component in the new design has the same name as a component in the guide design, that component matches the guide component.
- If a signal in the new design has the same name as a signal in the guide design, the signal matches the guide signal.

- Any matching component in the new design is placed in the site corresponding to the location of the matching guide component, if possible.
- Matching component pins are swapped to match those of the guide component with regard to matching signals, if possible.
- All of the connections between matching driver and load pins of the matching signals have the routing information preserved from the guide file with the exception of Vcc and GND signals.

When PAR runs using a guide design as input, PAR first places and routes any components and signals that fulfill the matching criteria described above. Then PAR places and routes the remainder of the logic.

To place and route the remainder of the logic, PAR performs the following:

- If you have selected exact guided PAR (by entering the **–gm exact** option on the PAR command line), the placement and routing of the matching logic are locked. Neither placement nor routing can be changed to accommodate additional logic.
- If you have selected leveraged guided PAR (by entering the –**gm leverage** option on the PAR command line), PAR tries to maintain the placement and routing of the matching logic, but changes placement or routing if it is necessary in order to place and route to completion and achieve your timing constraints (if possible).
- If you have selected incremental guided PAR (by entering the **-gm incremental** option on the PAR command line), your design must have area groups constraints to take advantage of this option. If an area group has changed, for example, additional or elimination of logic, this area group will not be guided. The other area groups will maintain the placement but routing will change to route the design completely and to achieve your timing constraints (if possible).

  Some cases where the leveraged mode is necessary are as follows:

  - You have added logic that makes it impossible to meet your timing constraints without changing the placement and routing in the guide design.
  - You have added logic that demands a certain site or certain routing resource, and that site or routing resource is already being used in the guide design.

Note: For Verilog or VHDL netlist designs, re-synthesizing modules typically causes signal and instance names in the resulting netlist to be significantly different from the netlist obtained in earlier synthesis runs. This occurs even if the source level Verilog or VHDL code only contains a small change. Because guided PAR depends on signal and component names, synthesis designs often have a low *match rate* when guided. Therefore, guided PAR is not recommended for most synthesis-based designs, although there may be cases where it could be a successful alternative technique.

PCI Cores

You can use a guide file to add a PCI Core, which is a standard I/O interface, to your design. The PCI Core guide file must already be placed and routed. PAR only places and routes the signals that run from the PCI Core to the input NCD design; it does not place or route any portion of the PCI Core. You can also use the resulting design (PCI Core integrated with your initial design) as a guide file. However, you must then use the **exact** option for –gm, *not* **leverage**, when generating a modified design.

Guided PAR supports precise matching of placement and routing of PCI Cores that are used as reference designs in a guide file:

- Components locked in the input design are guided by components in the reference design of a guide file in the corresponding location.

- Signals that differ only by additional loads in the input design have the corresponding pins routed according to the reference design in the guide file.
- Guide summary information in the PAR report describes the amount of logic from the reference design that matches logic in the input design.

For detailed information about designing with PCI Cores, refer to the Xilinx PCI web page at http://www.xilinx.com/systemio/pciexpress/index.htm.

PAR Syntax

The following syntax places and routes your design:

par [*options*] *infile*[**.ncd**] *outfile* [*pcf_file*[**.pcf**]]

options can be any number of the PAR options listed in "PAR Options." They do not need to be listed in any particular order. Separate multiple options with spaces.

infile is the design file you wish to place and route. The file must include a .ncd extension, but you do not have to specify the .ncd extension on the command line.

outfile is the target design file that is written after PAR is finished. If the command options you specify yield a single output design file, *outfile* has an extension of .ncd or .dir. A .ncd extension generates an output file in NCD format, and the .dir extension directs PAR to create a directory in which to place the output file (in NCD format). If the specified command options yield more than one output design file, *outfile* must have an extension of .dir. The multiple output files are placed in the directory with the .dir extension.

If the file or directory you specify already exists, an error messages appears and the operation is not run. You can override this protection and automatically overwrite existing files by using the –w option.

pcf_file is a Physical Constraints File (PCF). The file contains the constraints you entered during design entry, constraints you added using the User Constraints File (UCF) and constraints you added directly in the PCF file. If you do not enter the name of a PCF on the command line and the current directory contains an existing PCF with the *infile* name and a .pcf extension, PAR uses the existing PCF.

PAR Input Files

Input to PAR consists of the following files:

- NCD file—a mapped design file.
- PCF —an ASCII file containing constraints based on timing, physical placements, and other attributes placed in a UCF or NCF file. A list of constraints is located in the *Constraints Guide*. PAR supports all of the timing constraints described in the *Constraints Guide*.
- Guide NCD file—an optional placed and routed NCD file you can use as a guide for placing and routing the design.

PAR Output Files

Output from PAR consists of the following files:

- NCD file—a placed and routed design file (may contain placement and routing information in varying degrees of completion).

- PAR file—a PAR report including summary information of all placement and routing iterations.
- PAD file—a file containing I/O pin assignments in a parsable database format.
- CSV file—a file containing I/O pin assignments in a format supported by spreadsheet programs.
- TXT file—a file containing I/O pin assignments in a ASCII text version for viewing in a text editor.
- GRF (Guide Report File)— a file that is created when you use the –gf option.

PAR Options

You can customize the placement and routing of your design by specifying one or more of the command line options when you run PAR. You can place a design without routing it, perform a single placement, perform a number of placements using different cost tables, and specify an effort level (std, med, high) based on the complexity of your design.

The following tables list a summary of the PAR command line options, along with a short description of their functions, default settings, and effort levels:

Table 9-1: **Effort Level Options**

Option	Function	Range	Default
–ol *effort_level*	Overall placement and routing effort level	std,med, high	std
–pl *placer_effort_level*	Placement effort level (overrides the –ol value for the placer)	std,med, high	Determined by the –ol setting
–rl *router_effort_level*	Routing effort level (overrides **–ol value for the router)**	std,med, high	Determined by the –ol setting
–xe *extra_effort_level*	Set extra effort level (only available if -ol is set to high)	normal, continue	No extra effort level is used

Table 9-2: **General Options**

Option	Function	Range	Default
–f *command_file*	Executes command line arguments in a specified command file	N/A	No command line file
–intstyle	Reduced screen output to error and warning messages based on the integration style you are running	ise, xflow, silent	Display all information on the screen
–k	Run re-entrant router starting with existing placement and routing	N/A	Run placement and standard router (Do not run re-entrant routing)

Table 9-2: **General Options**

Option	Function	Range	Default
–nopad	Suppresses the creation of the PAD files in all three formats	N/A	Generate all PAD files
–p	Do not run the Placer	N/A	Run Placement
–power	Power Aware PAR	N/A	Off
–r	Do not run the Router	N/A	Run Router
–ub	Use bonded IOB sites for unbonded IOBs	N/A	Do not use bonded IOB sites
–w *existing_file*	Overwrite existing output files that have the same name and path	N/A	Do not overwrite
–x	Ignore any timing constraints provided and generate new timing constraints on all internal clocks. Adjust these constraints while PAR is running to focus on either performance or run time based on effort level setting.	N/A	Use timing constraints if present or use Automatic Timespecing if no timing constraints are given

Table 9-3: **Guide Options**

Option	Function	Range	Default
–gf	Specifies the name of a NCD file to be used as the guide file for PAR	N/A	No guide file is used
–gm	Selects the mode of guide to use during Place and Route	exact, leverage, incremental	Exact

Table 9-4: **Multi Pass Place and Route (MPPR) Options**

Option	Function	Range	Default
–n *iteration*	Number of Placement Cost Tables to run in Multi Pass Place and Route ***Note:*** When the value is set to 0, PAR runs up to all 100 cost tables until one meets timing.	0-100	One (1) place and route run
–m *nodefile_name*	Turns engine for Multi Pass Place and Route	N/A	Do not run the turns engine
–s *number_to_save*	Save number of results from Multi Pass Place and Route (for use with the –n option)	1-100	Saves all
–t *placer_cost_table*	Starting Placement Cost Table	1-100	One (Start placer at Cost Table 1)

Detailed Listing of Options

This section describes PAR options in more detail. The listing is in alphabetical order.

–f (Execute Commands File)

–f *command_file*

The –f option executes the command line arguments in the specified *command_file*. For more information on the –f option, see "–f (Execute Commands File)" in Chapter 1.

–gf (Guide NCD File)

–gf *guide_file*

The –gf option specifies the name of an NCD file (from a previous PAR run) to be used as a guide for the current PAR run. The guide file is an NCD file which is used as a template for placing and routing the input design. If the –gm option is not specified, the guide mode will be **exact**. For more information on the guide file, see "Guided PAR".

Note: Support for using multiple guide files is being deprecated and will not be available in future releases of Xilinx software.

–gm (Guide Mode)

–gm {exact | leverage | incremental}

The –gm option specifies the type of guided placement and routing PAR uses—exact, leverage, or incremental. The default is exact mode. For more information on the guide modes, see "Guided PAR".

Note: Support for the leverage guide flow (–gm incremental), without a timing-driven map run of your design, specified with the map –timing option, will not be supported in future releases of Xilinx software.

You must specify the NCD to use as a guide file by entering a –gf option (see "–gf (Guide NCD File)") on the PAR command line.

```
par -gf previous_run.ncd -gm leverage design_ncd place_and_routed.ncd
design.pcf
```

Note: The Incremental Design flow is being deprecated and will not be available in future releases of Xilinx software. New in 8.2i are Partitions, which provide significant flexibility and functionality for design preservation. Information on Partitions can be found in the online help included in the 8.2i software and in the "TCL chapter" of this book. Incremental Design using the map and par -gm incremental option will still work in 8.2i, though it generates a warning that this flow is being removed.

–intstyle (Integration Style)

-intstyle {**ise** | **xflow** | **silent**}

The –intstyle option reduces screen output based on the integration style you are running. When using the –intstyle option, one of three modes must be specified: *ise*, *xflow*, or *silent*. The mode sets the way information is displayed in the following ways:

-intstyle ise

This mode indicates the program is being run as part of an integrated design environment.

-intstyle xflow

This mode indicates the program is being run as part of an integrated batch flow.

-intstyle silent

This mode limits screen output to warning and error messages only.

Note: The -intstyle option is automatically invoked when running in an integrated environment, such as Project Navigator or XFLOW.

–k (Re-Entrant Routing)

-k *previous_NCD.ncd reentrant.ncd*

The –k option runs re-entrant routing. Routing begins with the existing placement and routing left in place. Re-entrant routing is useful to manually route parts of a design and then continue automatic routing, if you halted the route prematurely (for example, with Ctrl+C) and want to resume, or if you want to run additional route passes.

–m (Multi-Tasking Mode)

-m *nodefile_name*

The –m option allows you to specify the nodes on which to run jobs when using the PAR Turns Engine. You must use this option with the -n (Number of PAR Iterations) option.

```
par -m nodefile_name -ol high -n 10 mydesign.ncd output.dir
```

Note: The –m option is not supported on Windows operating systems.

–n (Number of PAR Iterations)

–n *iterations*

By default (without the –n option), one place and route iteration is run. The –n option determines the number of place and route passes performed at the effort level specified by the –ol option. Each iteration uses a different cost table when the design is placed and produces a different NCD file. If you enter -n 0, the software continues to place and route, stopping only when the design is fully routed and meets all timing constraints, or after completing the iteration of cost table 100. If you specify a –t option, the iterations begin at the cost table specified by –t. The valid range of the cost table is 0–100; default is 1.

```
par -pl high -rl std -n 5 design.ncd output.dir design.pcf
```

Note: For the best MPPR results in a reasonable time, use –pl high –rl std to get the best placement.

–nopad (No Pad)

-nopad

The –nopad option turns off the generation of the three output formats for the PAD file report. By default, all three report types are created when PAR is run.

–ol (Overall Effort Level)

–ol *effort_level*

The –ol option sets the overall PAR effort level. The effort level specifies the level of effort PAR uses to place and route your design to completion and to achieve your timing constraints.

Of the three *effort_level* values, use std on the least complex design, and high on the most complex. The level is not an absolute; it shows instead relative effort.

If you place and route a simple design at a complex level, the design is placed and routed properly, but the process takes more time than placing and routing at a simpler level. If you place and route a complex design at a simple level, the design may not route to completion or may route less completely (or with worse delay characteristics) than at a more complex level.

Increasing your overall level will enable harder timing goals to be possibly met, however it will increase your runtime.

The *effort_level* setting is std, med, or high with the default level std.

The –ol level sets an effort level for placement and another effort level for routing. These levels are also std, med, high. The placement and routing levels set at a given –ol level depend on the device family in the NCD file. You can determine the default placer and router effort levels for a device family by reading the PAR Report file produced by your PAR run.

You can override the placer level set by the –ol option by entering a –pl (Placer Effort Level) option, and you can override the router level by entering a –rl (Router Effort Level) option.

```
par -ol high design.ncd output.ncd design.pcf
```

–p (No Placement)

The –p option bypasses the placer and proceeds to the routing phase. A design must be fully placed when using this option or PAR will issue an error message and exit. When you use this option, existing routes are ripped up before routing begins. You can, however, leave the routing in place if you use the –k option instead of the –p option.

```
par -p design.ncd output.ncd design.pcf
```

Note: The –p option is recommended when you have an MFP or UCF written from Floorplanner or wish to maintain a previous NCD placement but run the router again.

–pl (Placer Effort Level)

–pl *placer_effort_level*

The –pl option sets the placer effort level. The effort level specifies the level of effort used when placing the design. This option overrides the setting specified for the –ol option. For a description of effort level, see "–ol (Overall Effort Level)".

The *placer_effort_level* setting is std, med, or high, and the default level set if you do not enter a –pl option is determined by the setting of the –ol option.

```
par -pl high placed_design.ncd output.ncd design.pcf
```

–power (Power Aware PAR)

The –power option optimizes the capacitance of non-timing driven design signals. The default setting for this option is *off*.

–r (No Routing)

Use the –r option to prevent the routing of a design. The –r option causes the design to exit before the routing stage.

```
par -r design.ncd route.ncd design.pcf
```

–rl (Router Effort Level)

–rl *router_effort_level*

The –rl option sets the router effort level. The effort level specifies the level of effort used when routing the design. This option overrides the setting for the –ol option. For a description of effort level, see "–ol (Overall Effort Level)".

The *router_effort_level* setting is std, med, or high, and the default level set if you do not enter a –rl option is determined by the setting of the –ol option. In the example that follows, the placement level is at std (default) and the router level is at the highest effort level.

```
par -rl high design.ncd output.ncd design.pcf
```

–s (Number of Results to Save)

–s *number_to_save* **–n** *iterations*

The –s option is used with the –n option to save the number of results that you specify. By default (with no –s option), all results are saved. The valid range for the *number_to_save* is 1–100.

The –s option compares the results of each iteration and saves the best output NCD files. The best NCDs are determined by a score assigned to each output design. This score takes into account such factors as the number of unrouted nets, the delays on nets and conformance to any timing constraints. The lower the score, the better the design. This score is described in the PAR Reports section of this chapter. See the Multi Pass Place and Route (MPPR) section for more details.

```
par -s 2 -n 10 -pl high -rl std design.ncd output_directory design.pcf
```

–t (Starting Placer Cost Table)

–t *placer_cost_table*

When the -n option is specified without the -t option, PAR starts at placer cost table 1.The –t option specifies the cost table at which the placer starts (placer cost tables are described in "Placing"). If the cost table 100 is reached, placement begins at 1 again, if you are running MPPR due to the –n options. The *placer_cost_table range* is 1–100, and the default is 1.

```
par –t 10 –s 1 –n 5 –pl high –rl std design.ncd output_directory
   design.pcf
```

The previous option is often used with MPPR to try out various cost tables. In this example, cost table 10 is used and a MPPR run is performed for 5 iterations. The par run starts with cost table 10 and runs through 14. The placer effort is at the highest and the router effort at std. The number of NCD saved will be the best one.

Note: See the Multi Pass Place and Route (MPPR) section for more details.

–ub (Use Bonded I/Os)

```
par –ub design.ncd output.ncd design.pcf
```

By default (without the -ub option), I/O logic that MAP has identified as internal can only be placed in unbonded I/O sites. If the –ub option is specified, PAR can place this internal I/O logic into bonded I/O sites in which the I/O pad is not used. The option also allows PAR to route through bonded I/O sites. If you use the –ub option, make sure this logic is not placed in bonded sites connected to external signals, power, or ground. You can prevent this condition by placing PROHIBIT constraints on the appropriate bonded I/O sites. See the *Constraints Guide* for more information on constraints.

–w (Overwrite Existing Files)

```
par input.ncd –w existing_NCD.ncd input.pcf
```

Use the –w option to instruct PAR to overwrite existing output files, including the input design file if it follows the –w option. The default is not to overwrite an NCD. Therefore if the given NCD exists, then PAR gives an error and terminates before running place and route.

–x (Performance Evaluation Mode)

```
par –x design.ncd output.ncd design.pcf
```

The -x option is used if there are timing constraints specified in the physical constraints file, and you want to execute a PAR run with tool-generated timing constraints instead to evaluating the performance of each clock in the design. This operation is referred to as "Performance Evaluation" mode. This mode is entered into either by using the -x option or when no timing constraints are used in a design.The tool-generated timing constraints constrain each internal clock separately and tighten/loosen the constraints based on feedback during execution. The PAR effort level controls whether the focus is on fastest run time (STD), best performance (HIGH) or a balance between run time and performance (MED).

PAR ignores all timing constraints in the design.pcf, and uses all physical constraints, such as LOC and AREA_RANGE.

–xe (Extra Effort Level)

–xe *effort_level*

```
par –ol high –xe n design.ncd output.ncd design.pcf
```

Use the –xe option to set the extra effort level. The *effort_level* variable can be set to *n* (normal) or *c* (continue) working even when timing cannot be met. Extra effort *n* uses additional runtime intensive methods in an attempt to meet difficult timing constraints. If PAR determines that the timing constraints cannot be met, then a message is issued explaining that the timing cannot be met and PAR exits. Extra effort *c* allows you to direct PAR to continue routing even if PAR determines the timing constraints cannot be met. PAR continues to attempt to route and improve timing until little or no timing improvement can be made.

Note: Use of extra effort *c* can result in extremely long runtimes.

PAR Reports

The output of PAR is a placed and routed NCD file (the output design file). In addition to the output design file, a PAR run generates a report file with a .par extension. A Guide Report File (.grf) is created when you use the –gf option.

Note: The ReportGen utility can be used to generate pad report files (.pad, pad.txt, and pad.csv). The pinout .pad file is intended for parsing by user scripts. The pad.txt file is intended for user viewing in a text editor. The pad.csv file is intended for directed opening inside of a spreadsheet program. It is not intended for viewing through a text editor. See the "ReportGen" section of this chapter for information on generating and customizing pad reports.

The PAR report contains execution information about the place and route job as well as all constraint messages.

If the options that you specify when running PAR are options that produce a single output design file, your output is the output design (NCD) file, a PAR file, and PAD files. The PAR file and the PAD files have the same root name as the output design file.

If you run multiple iterations of placement and routing, you produce an output design file, a PAR file, and PAD files for each iteration. Consequently, when you run multiple iterations you have to specify a directory in which to place these files.

As the command is performed, PAR records a summary of all placement and routing iterations in one PAR file at the same level as the directory you specified, then places the output files (in NCD format) in the specified directory. Also, a Place and Route Report File and a PAD file are created for each NCD file, describing in detail each individual iteration.

Note: Reports are formatted for viewing in a monospace (non-proportional) font. If the text editor you use for viewing the report uses a proportional font, the columns in the report do not line up correctly. The pad.csv report is formatted for importing into a spreadsheet program or for parsing via a user script.

Place and Route Report File

The Place and Route report file contains execution information about the PAR command run. The report file shows the steps taken as the program converges on a placement and routing solution. A sample PAR report file follows:

The first lines of the PAR report identify the software version you are running, the machine on which it is run, and the date and time stamp. In addition, the command line entry is restated, along with information about the input design files (NCD and PCF). Warning messages also appear in the first section of the PAR report.

```
Release 8.1i - par HEAD
Copyright (c) 1995-2005 Xilinx, Inc.  All rights reserved.

Par -w -ol high c:\test\test_top_mod_map.ncd c:\test\par0.ncd
c:\test\test_top_mod.pcf

Constraints file: c:\test\test_top_mod.pcf.
Loading device for application Rf_Device from file '2vp2.nph' in envi-
ronment c:/Xilinx.
   "test_top_mod" is an NCD, version 1.0, device xc2vp2, package ff672,
   speed -7

Initializing temperature to 100.000 Celsius. (default - Range: -40.000
to 100.000 Celsius)
Initializing voltage to 1.500 Volts. (default - Range: 1.400 to 1.600
Volts)

WARNING:Timing:2796 - The input clock clkB_IBUFG to DCM
   test_lutram_bram/test_DCM has a period (frequency) specification of
   2700 ps (370.37 Mhz). This violates the minimum period (maximum fre-
   quency) of 4761 ps (210.04 Mhz).
WARNING:Timing:2798 - The output clock test_lutram_bram/CLK0_W from DCM
   test_lutram_bram/test_DCM has a period (frequency) specification of
   2700 ps (370.37 Mhz).  This violates the minimum period (maximum fre-
   quency) of 4761 ps (210.04 Mhz).
```

The next section of the PAR report provides a breakdown of the resources in the design and includes the Device Utilization Summary.

```
Device speed data version:  "PRODUCTION 1.90 2005-12-13".

Device Utilization Summary:

   Number of BUFGMUXs                  4 out of 16     25%
   Number of DCMs                      1 out of 4      25%
   Number of External IOBs            80 out of 204    39%
```

```
      Number of LOCed IOBs            78 out of 80     97%

   Number of RAMB16s                   1 out of 12      8%
   Number of SLICEs                   26 out of 1408    1%

Overall effort level (-ol):   High (set by user)
Placer effort level (-pl):    High (set by user)
Placer cost table entry (-t): 1
Router effort level (-rl):    High (set by user)

Starting initial Timing Analysis.  REAL time: 31 secs
Finished initial Timing Analysis.  REAL time: 31 secs
```

As shown in the next section, PAR reports different phases of the placer and identifies which phase is being executed. The checksum number shown is for Xilinx debugging purposes only and does not reflect the quality of the placer run. A running tally of the time transpired since starting PAR is also shown in this section of the PAR report.

```
Starting Placer

Phase 1.1
Phase 1.1 (Checksum:989a1f) REAL time: 43 secs

Phase 2.31
Phase 2.31 (Checksum:1312cfe) REAL time: 43 secs

Phase 3.2
Phase 3.2 (Checksum:1c9c37d) REAL time: 47 secs

Phase 4.30
Phase 4.30 (Checksum:26259fc) REAL time: 47 secs

Phase 5.3
Phase 5.3 (Checksum:2faf07b) REAL time: 47 secs

Phase 6.5
Phase 6.5 (Checksum:39386fa) REAL time: 47 secs

Phase 7.8
Phase 7.8 (Checksum:9a609d) REAL time: 47 secs

Phase 8.5
Phase 8.5 (Checksum:4c4b3f8) REAL time: 47 secs

Phase 9.18
Phase 9.18 (Checksum:55d4a77) REAL time: 47 secs

Phase 10.24
Phase 10.24 (Checksum:5f5e0f6) REAL time: 47 secs

Phase 11.27
Phase 11.27 (Checksum:68e7775) REAL time: 47 secs
```

```
Phase 12.5
Phase 12.5 (Checksum:7270df4) REAL time: 47 secs

Writing design to file c:\test\par0.ncd

Total REAL time to Placer completion: 47 secs
Total CPU time to Placer completion: 8 secs
```

The router portion of the PAR report shows that the router has been invoked. It displays each phase of the router and reports the number of unrouted nets, in addition to an approximate timing score in parenthesis.

```
Starting Router

Phase 1: 231 unrouted;       REAL time: 51 secs

Phase 2: 154 unrouted;       REAL time: 51 secs

Phase 3: 21 unrouted;       REAL time: 51 secs

Phase 4: 21 unrouted; (347)      REAL time: 51 secs

Phase 5: 21 unrouted; (0)      REAL time: 51 secs

Phase 6: 21 unrouted; (0)      REAL time: 51 secs

Phase 7: 0 unrouted; (0)      REAL time: 51 secs

Total REAL time to Router completion: 51 secs
Total CPU time to Router completion: 10 secs

Generating "par" statistics.
```

The next portion of the PAR report contains the Clock Report, which includes a clock table that lists all clocks in the design and provides information on the routing resources, number of fanout, maximum net skew for each clock, and maximum delay. The Locked column in the clock table means the clock driver (BUFGMUX) is assigned to a particular site instead of left floating.

Note: The clock skew and delay listed in this table differ from the skew and delay reported in TRACE, or Timing Analyzer. PAR takes into account the net that drives the clock pins whereas TRACE and Timing Analyzer include the entire clock path.

```
*************************
Generating Clock Report
*************************

+---------------------+--------------+------+------+------------+----------+
|        Clock Net    |   Resource   |Locked|Fanout|Net Skew(ns)|Max Delay(ns)
+---------------------+--------------+------+------+------------+----------+
|test_lutram_bram/CLK |              |      |      |            |
|                  1X |     BUFGMUX1P| No   |   43 |  0.023     |  0.724
+---------------------+--------------+------+------+------------+----------+
|      clk_bram_BUFGP |     BUFGMUX6S| No   |    6 |  0.004     |  0.704
+---------------------+--------------+------+------+------------+----------+
|     clk_slice_BUFGP |     BUFGMUX3P| No   |    4 |  0.000     |  0.700
+---------------------+--------------+------+------+------------+----------+
|  clk_lut_test_BUFGP |     BUFGMUX7P| No   |   11 |  0.027     |  0.715
+---------------------+--------------+------+------+------------+----------+
```

The next portion of the PAR report lists information on timing constraints contained in the input PCF. The first line of this section shows the Timing Score, which in this example is 0. In cases where a timing constraint is not met, the Timing Score will be greater than 0. The lower the Timing Score, the better the result.

Note: The constraints table in this section is not generated when no constraints are given in the input PCF, or the –x option is used.

```
Timing Score: 0

Asterisk (*) preceding a constraint indicates it was not met.
   This may be due to a setup or hold violation.
----------------------------------------------------------------------------
  Constraint                                | Requested  | Actual     | Logic
                                            |            |            | Levels
----------------------------------------------------------------------------
  TS_clk_lut_test = PERIOD TIMEGRP "clk_lut | 2.800ns    | 2.761ns    | 1
  _test"  2.800 nS   HIGH 50.000000 %       |            |            |
----------------------------------------------------------------------------
  TS_clk_bram = PERIOD TIMEGRP "clk_bram"   | 1.400ns    | 1.339ns    | 1
  1.400 nS   HIGH 50.000000 %               |            |            |
----------------------------------------------------------------------------
  TS_clk_slice = PERIOD TIMEGRP "clk_slice" | 1.200ns    | 1.150ns    | 1
    1.200 nS   HIGH 50.000000 %             |            |            |
----------------------------------------------------------------------------
  TS_clkB = PERIOD TIMEGRP "clkB"  2.700 nS | N/A        | N/A        | N/A
     HIGH 50.000000 %                       |            |            |
----------------------------------------------------------------------------
  TS_test_lutram_bram_CLK0_W = PERIOD TIMEG | 2.700ns    | 2.494ns    | 0
  RP "test_lutram_bram_CLK0_W" TS_clkB *  1 |            |            |
  .000000 HIGH 50.000 %                     |            |            |
----------------------------------------------------------------------------
  OFFSET = IN 600 pS  BEFORE COMP "clk_lut_ | 0.600ns    | 0.565ns    | 1
  test" TIMEGRP "Lut_test_1_and_2"          |            |            |
----------------------------------------------------------------------------
```

The last portion of the PAR report shows how many timing constraints were met and whether PAR was able to place and route the design successfully. The total time used to complete the PAR run is displayed in both REAL time and CPU time. Any errors and a message summary are also shown.

```
All constraints were met.
INFO:Timing:2761 - N/A entries in the Constraints list may indicate that the
   constraint does not cover any paths or that it has no requested value.
Generating Pad Report.

All signals are completely routed.

Total REAL time to PAR completion: 53 secs
Total CPU time to PAR completion: 11 secs

Peak Memory Usage:  99 MB

Placement: Completed - No errors found.
Routing: Completed - No errors found.
Timing: Completed - No errors found.

Number of error messages: 0
Number of warning messages: 2
Number of info messages: 0

Writing design to file c:\test\par0.ncd
par done!
```

Multi Pass Place and Route (MPPR)

When running multiple iterations of the placer and router, PAR produces output design files for each iteration. When you run multiple iterations, you must specify a directory for PAR to place these files. PAR records a summary of all placement and routing iterations in one PAR file at the same level as the directory that you specify. Then PAR places the output design files, in NCD format, in the specified directory. For each NCD file, a PAR and PAD files (CSV, TXT, PAD) are also created, describing in detail each individual iteration.

Note: The naming convention for the output files, which may contain placement and routing information in varying degrees of completion, is:

[placer effort level]_[router effort level]_[cost table number]

The following is a command line example for running three iterations of the placer and router, using a different cost table entry each time.

```
par -n 3 -pl high -rl std address.ncd output.dir
```

–n 3 is the number of iterations you want to run,

–pl high sets the placement effort level to high

–rl std sets the router effort level

address.ncd is the input design file

output.dir is the name of the directory in which you want to place the results of the PAR run.

Note: Cost table 1, the default, is used for the initial cost table because no initial cost table was specified.

One strategy for using MPPR is to set the placer effort level to *high* and router effort level to *std* to ensure a quality placement and a quick route. This strategy enables PAR to run the cost tables effectively and reduces the total runtime of all place and route iterations.

When a design is very close to reaching its timing goals and can run for a long period through all the cost tables, another strategy is to use the following:

```
par –n 0 -ol high, -xe n address.ncd output.dir
```

Select I/O Utilization and Usage Summary

If more than one Select I/O standard is used, an additional section on Select I/O utilization and usage summary is added to the PAR file. This section shows details for the different I/O banks. It shows the I/O standard, the output reference voltage (VCCO) for the bank, the input reference voltage (VREF) for the bank, the PAD and Pin names. In addition this section gives a summary for each bank with the number of pads being used, the voltages of the VREFs, and the VCCOs.

Importing the PAD File Information

The PAD (pad and _pad.csv) reports are formatted for importing into a spreadsheet program such as Microsoft® Excel, or for parsing via a user script. The _pad.csv file can be directly opened by Microsoft Excel. The procedure for importing a .pad file into Microsoft Excel is as follows:

1. In Excel, select the menu **File** → **Open**.
2. In the Open dialog box, change the Files of type field to **All Files** (*.*)" Browse to the directory containing your .pad file. Select the file so it appears in the File name field. Select the **Open** button to **close** the Open dialog box.
3. The Excel Text Import Wizard dialog appears. In the Original data type group box, select **Delimited**. Select the **Next** button to proceed.
4. In the Delimiters group box, uncheck the **Tab** checkbox. Place a **check** next to Other and enter a | character into the field after Other. The | symbol is located on the keyboard above the *Enter* key.
5. Select the **Finish** button to complete the process.

You can then format, sort, print, etc. the information from the PAD file using spreadsheet capabilities as required to produce the necessary information.

Note: This file is designed to be imported into a spreadsheet program such as Microsoft Excel for viewing, printing, and sorting.The "|" character is used as the data field separator. This file is also designed to support parsing.

Guide Reporting

For guided PAR, the PAR report displays summary information describing the total amount and percentage of components and signals in the input design guided by the reference design. The report also displays the total/percentage of components and signals from the reference design (guide file) that were used to guide the input design.

The guide report, which is included in the PAR report file, is generated with the -gf option. The report describes the criteria used to select each component and signal used to guide the design. It may also enumerate the criteria used to reject some subset of the components and signals that were eliminated as candidates. See the Guided PAR section of this chapter for more information on using guide files.

Xplorer

Xplorer is a TCL script that seeks the best design performance using ISE implementation software. After synthesis generates an EDIF or NGC (XST) file, the design is ready for implementation. During this phase, you can use Project Navigator or the command line to manually apply design constraints and explore different implementation tool settings for achieving your timing goals; alternatively, you can use Xplorer. Xplorer is designed to help achieve optimal results by employing smart constraining techniques and various physical optimization strategies. Because no unique set of ISE options or timing constraints works best on all designs, Xplorer finds the right set of implementation tool options to either meet design constraints or find the best performance for the design. Hence, Xplorer has two modes of operation: Best Performance Mode and Timing Closure Mode.

Xplorer support is available for the following Xilinx FPGA architectures:

- Virtex™-II Pro, Virtex™-II Pro X
- Virtex™-4 /FX/LX/SX
- Virtex™-5 LX
- Spartan™-3, Spartan™-3E, Spartan™-3L

Best Performance Mode

In this mode, Xplorer optimizes design performance for a user-specified clock domain, allowing easy evaluation of the maximum achievable performance. You specify the design name and a single clock to optimize. Xplorer implements the design with different architecture-specific optimization strategies in conjunction with timing-driven place and route (PAR). When the -clk option is specified, it tightens or relaxes the timing constraints depending on whether or not the frequency goal is achieved. Xplorer estimates the starting frequency based on pre-PAR timing data. Adjusting timing constraints such that PAR is neither under nor over-constrained, enables Xplorer to deliver optimal design performance.

In addition to timing constraints, Xplorer also uses physical optimization strategies such as global optimization and timing-driven packing and placement. Global optimization performs pre-placement netlist optimizations on the critical region, while timing-driven packing and placement provides closed-loop packing and placement such that the placer can recommend logic packing techniques that deliver optimal placement. If the design has a User Constraints File (UCF), Xplorer optimizes for the user constraints on the specified clock domain.

Following is sample command line syntax for Best Performance Mode. For a complete list of Xplorer options, see "Xplorer Options."

Example:	**xplorer.tcl <*design_name*> -clk <*clock_name*> –p <*part_name*>**
Description:	*design_name* specifies the name of the top-level EDIF or NGC file. *clock name*, specified with the -clk option, specifies the name of the clock to optimize. *part_name*, specified with the -p option, specifies the Xilinx part name.

Result:	Xplorer implements the design with different architecture-specific optimization strategies and timing-driven placement and routing for optimal design performance. The results are summarized in the output Xplorer report (.rpt). The best run is identified at the end of the report file.

Timing Closure Mode

If you have a design with timing constraints and your intent is for the tools to meet the specified constraints, use the Timing Closure Mode. In this mode, you should not specify a clock with the -clk option. Xplorer looks at the UCF to examine the goals for timing constraints. Using these constraints together with optimization strategies such as global optimization, timing-driven packing and placement, register duplication, and cost tables, Xplorer implements the design in multiple ways to deliver optimal design performance.

Because Xplorer runs approximately 10 iterations, you will experience longer PAR runtimes. However, Xplorer is something that is typically run once during a design cycle. After an Xplorer run, you can capture the set of options that will give the best result from the Xplorer report file and use that set of options for future design runs. Typically, designers run the implementation tools many times in a design cycle, so a longer initial runtime will likely reduce the number of PAR iterations later.

Following is sample command line syntax for Timing Closure Mode. For a complete list of Xplorer options, see "Xplorer Options.":

Example:	**`xplorer.tcl <design_name> -uc <ucf_name> -p <part_name>`**
Description:	*design_name* specifies the name of the top-level EDIF or NGC file. *ucf_name*, specified with the -uc option, specifies the name of the UCF that Xplorer uses to examine the goals for timing constraints. *part_name*, specified with the -p option, specifies the complete Xilinx part name.
Result:	Xplorer implements the design in multiple ways to deliver optimal design performance. The results are summarized in the output Xplorer report (.rpt). The best run is identified at the end of the report file.

Xplorer Syntax

Xplorer is run from the command line or from the Project Navigator GUI. For information on running Xplorer in Project Navigator, please see the "Using Xplorer in ISE" topic in the online help included with the ISE software. The following syntax is used to run Xplorer script from the command line. Note that when using Windows, the .tcl extension is not used.

```
xplorer.tcl <design_name> [options] -p <part_name>
```

design_name is the name of the top-level EDIF or NGC file. This should be the simple name of the file, rather than a full absolute or relative path. If the file is not in the current directory, use the –sd and –wd options to specify the directory where the design resides and the directory in which to write any output files.

options can be any number of the Xplorer options listed in "Xplorer Options." Use the -clk option for Best Performance Mode and the -uc option for Timing Closure Mode. Separate multiple options with spaces.

part_name is the complete name of the Xilinx part, specified with the –p option.

Note: You can also run Xplorer from the Xilinx TCL Shell, accessible through the TCL Console tab in Project Navigator.

Xplorer Input Files

Input to Xplorer consists of the following files:

- EDIF—netlist file produced by synthesis.
- NGC—netlist file produced by XST.

Xplorer Output Files

Output from Xplorer consists of following:

RPT—report file that summarizes all of the results from the Xplorer runs. The best run is identified at the end of the report file. See "Xplorer Report" in this chapter for additional information.

LOG—log files that contain all standard out messages. Xplorer produces two log files: xplorer.log and run.log.

run<*1*>.*—multiple log files with execution details for each Xplorer run. File names are based on the individual Xplorer run; for example, run1.log, run1.ucf, run 2.log, run2.ucf.

Xplorer Options

The following table lists the Xplorer command line options, along with a short description of each option.

Table 9-5: **Xplorer Options**

Option	Function
–bm <*bmm_file_name*>	Specifies the BMM file name for block RAM initialization.
–clk <*clock_name*>	Specifies the name of the clock net you wish to optimize in Best Performance Mode. If the –clk option is omitted, the script will use the timespec defined in the User Constraints File (UCF).
–freq <*value_in_MHz*>	Specifies the first attempted frequency in Best Performance Mode. The starting value impacts the number of runs. Without this option, the script determines a good starting value.
–map_options <*option(s)*>	Specifies additional MAP options to use during the Xplorer runs. See "MAP Options" in Chapter 7 for a complete list of MAP options. Separate multiple options with spaces.

Table 9-5: **Xplorer Options**

Option	Function
–max_runs <*number*>	Specifies the maximum number of implementation iterations that Xplorer runs on a design. Limiting the number of runs may adversely affect the final achieved frequency. The number of runs can be any number between 1 and 20. By default, this option is set to 7.
–no_retiming	Disables retiming in MAP. The default value for this option is to perform retiming.
–p <*part_name*>	Specifies the complete Xilinx part name. The default value is the part specified in the input design.
–sd <*directory_name*>	Specifies a list of paths for directories that contain the design files. The directories in the path list are separated by a comma (,). The first entry is the path to the location of the top-level design. The default value is the current directory.
–uc <*ucf_name*>	Specifies the name of the User Constraints File (UCF).
–wd <*directory_name*>	Specifies where the source and output files from the Xplorer runs will be stored and used, if the –sd option is not specified. The default value is the current directory.

Xplorer Report

The Xplorer report (.rpt) contains information on the Xplorer runs. It lists details for each run, including the options used by the implementation tools and the timing score. The best run is listed at the bottom of the Xplorer report.

The following is an example of an Xplorer report.

```
--------------------------------------------------------------------
FPGA Xplorer (tm) Version 2.39

2006-04-17 16:29:06

Command: xplorer system.ngc -uc=constraints.ucf
--------------------------------------------------------------------
 Run 1
--------------------------------------------------------------------
 Map options                          :  -timing -ol high -xe n
 Par options                          :  -w -ol high
 Achieved Timing Score                :  1002.00
 Current Best (Lowest) Timing Score   :  1002.00
 Current Best Run                     :  1
--------------------------------------------------------------------
```

```
------------------------------------------------------------------------
 Run 2
------------------------------------------------------------------------
 Map options                         :
 Par options                         :  -w -ol high -xe n
 Achieved Timing Score               :  1618.00
 Current Best (Lowest) Timing Score  :  1002.00
 Current Best Run                    :  1
------------------------------------------------------------------------
------------------------------------------------------------------------
 Run 3
------------------------------------------------------------------------
 Map options                         :  -timing -ol high -xe n
 Par options                         :  -w -ol high
 Achieved Timing Score               :  1002.00
 Current Best (Lowest) Timing Score  :  1002.00
 Current Best Run                    :  1
------------------------------------------------------------------------
------------------------------------------------------------------------
 Run 4
------------------------------------------------------------------------
 Map options                         :  -retiming on -timing -ol high -
xe n -global_opt on -ignore_keep_hierarchy
 Par options                         :  -w -ol high -t 9 -xe n
 Achieved Timing Score               :  1425.00
 Current Best (Lowest) Timing Score  :  1002.00
 Current Best Run                    :  1
------------------------------------------------------------------------
------------------------------------------------------------------------
 Run 5
------------------------------------------------------------------------
 Map options                         :  -cm balanced
 Par options                         :  -w -ol high
 Achieved Timing Score               :  13687.00
 Current Best (Lowest) Timing Score  :  1002.00
 Current Best Run                    :  1
------------------------------------------------------------------------
------------------------------------------------------------------------
 Run 6
------------------------------------------------------------------------
 Map options                         :  -timing -ol high -xe n -pr b -
cm balanced
 Par options                         :  -w -ol high
 Achieved Timing Score               :  2078.00
```

```
 Current Best (Lowest) Timing Score   :  1002.00
 Current Best Run                     :  1
--------------------------------------------------------------------
--------------------------------------------------------------------
 BestRun : Run 1
--------------------------------------------------------------------
 Map options                          :  -timing -ol high -xe n
 Par options                          :  -w -ol high
 Achieved Timing Score                :  1002.00
--------------------------------------------------------------------
```

ReportGen

The ReportGen utility generates reports specified on the command line. ReportGen takes an NCD file as input and outputs various pad reports and a log file that contains standard copyright and usage information on any reports being generated. Any reports generated by ReportGen must be specified on the command line using one or more of the ReportGen options. See , "ReportGen Options."

Note: Some reports require placed, and placed and routed NCD files as input.

ReportGen Syntax

The following syntax runs the reportgen utility:

reportgen [*options*] [*–pad [–padfmt pad | csv | txt] infile* [**.ncd**]]

options can be any number of the ReportGen options listed in the ReportGen Options section of this chapter. They do not need to be listed in any particular order. Separate multiple options with spaces.

pad is the pad report format you want to generate. By default ReportGen generates all format types, or you can use the –padfmt option to specify a specific format.

infile is the design file you wish to place and route. The file must include a .ncd extension, but you do not have to specify the .ncd extension on the command line.

ReportGen Input Files

Input to ReportGen consists of the following files:

- NCD file—a mapped design.

ReportGen Output Files

Output from ReportGen consists of the following report files:

- DLY file—a file containing delay information on each net of a design.
- PAD file—a file containing I/O pin assignments in a parsable database format.
- CSV file—a file containing I/O pin assignments in a format directly supported by spreadsheet programs.
- TXT file—a file containing I/O pin assignments in a ASCII text version for viewing in a text editor.

Files output by ReportGen are placed in the current working directory or the path that is specified on the command line with the -o option. ReportGen Options. The output pad files have the same root name as the output design file, but the .txt and .csv files have the tag "pad" added to the output design name. For example, *output_pad.txt*.

ReportGen Options

You can customize ReportGen output by specifying options when you run ReportGen from the command line. You must specify the reports you wish to generate.

The following table lists available ReportGen options and includes a usage example and functional description for each option

Option	Usage	Function
–f	reportgen –f *cmdfile*.cmd	Read ReportGen command line arguments and switches specified in a command file
–h	reportgen –h	Display reportgen usage information and help contents
–intstyle	reportgen –intstyle [*ise* | *xflow* | *silent*]	Reduce screen output to error and warning messages based on the integration style you are running
–o	reportgen –o	Specify the report output directory and filename
–pad	reportgen *design*.ncd –pad	Generate a pad report file
–padfmt {*pad* | *csv* | *txt*}	reportgen *design*.ncd –pad –padfmt *csv*	Generate a pad report in a specified format
–padcolsort	reportgen *design*.ncd –pad –padfmt [*pad* | *csv* | *txt*] –padsortcol *5,1:3*	Generate a specified pad report sorted on specified columns. Default: No sorting and all columns are displayed.
–r	reportgen *design*.ncd –r delay	Generate a delay report file in text format.

Turns Engine (PAR Multi-Tasking Option)

This Xilinx Development System option allows you to use multiple systems (nodes) that are networked together for a multi-run PAR job, significantly reducing the total amount of time to completion. You can specify this multi-tasking option from the command line.

Turns Engine Overview

Before the Turns Engine was developed for the Xilinx Development System, PAR could only run multiple jobs in a linear way. The total time required to complete PAR was equal to the sum of the times that it took for each of the PAR jobs to run. This is illustrated by the following PAR command.

```
par -ol high -n 10 mydesign.ncd output.dir
```

The preceding command tells PAR to run 10 place and route passes (**–n 10**) at effort level high (**–ol high**). PAR runs each of the 10 jobs consecutively, generating an output NCD file for each job, (i.e., output.dir/high_high_1.ncd, output.dir/high_high_2.ncd, etc.). If each job takes approximately one hour, then the run takes approximately 10 hours.

The Turns Engine allows you to use multiple nodes at the same time, dramatically reducing the time required for all ten jobs. To do this you must first generate a file containing a list of the node names, one per line. The following example shows five nodes for running 10 jobs.

Note: A pound sign (#) in the example indicates a comment.

```
# NODE names

jupiter             #Fred's node
mars                #Harry's node
mercury             #Betty's node
neptune             #Pam's node
pluto               #Mickey's node
```

Now run the job from the command line as follows:

par –m nodefile_name –ol high –n 10 mydesign.ncd output.dir

nodefile_name is the name of the node file you created.

This runs the following jobs on the nodes specified.

Table 9-6: **Node Files**

jupiter	par –ol high –i 10 –c 1 mydesign.ncd output.dir/high_high_1.ncd
mars	par –ol high –i 10 –c 1 mydesign.ncd output.dir/high_high_2.ncd
mercury	par –ol high –i 10 –c 1 mydesign.ncd output.dir/high_high_3.ncd
neptune	par –ol high –i 10 –c 1 mydesign.ncd output.dir/high_high_4.ncd
pluto	par –ol high –i 10 –c 1 mydesign.ncd output.dir/high_high_5.ncd

As the jobs finish, the remaining jobs are started on the five nodes until all 10 jobs are complete. Since each job takes approximately one hour, all 10 jobs complete in approximately two hours.

Note: You cannot determine the relative benefits of multiple placements by running the Turns Engine with options that generate multiple placements, but do not route any of the placed designs (the –r PAR option specifies *no routing*). The design score you receive is the same for each placement. To get some indication of the quality of the placed designs, run the route with a minimum router effort std (-rl std) in addition to the -ol high setting.

Turns Engine Syntax

The following is the PAR command line syntax to run the Turns Engine.

par –m *nodelist_file* **–n** *#_of_iterations* **–s** *#_of_iterations_to_save mapped_desgin***.ncd** *output_directory***.dir**

–m *nodelist_file* specifies the nodelist file for the Turns Engine run.

–n *#_of_iterations* specifies the number of place and route passes.

–s *#_of_iterations_to_save* saves only the best –s results.

*mapped design***.ncd** is the input NCD file.

*output_directory***.dir** is the directory where the best results (–s option) are saved. Files include placed and routed NCD, summary timing reports (DLY), pinout files (PAD), and log files (PAR).

Turns Engine Input Files

The following are the input files to the Turns Engine.

- NCD File—A mapped design.
- Nodelist file—A user-created ASCII file listing workstation names. The following is a sample nodelist file:

```
# This is a comment
# Note: machines are accessed by Turns Engine
# from top to bottom
# Sparc 20 machines running Solaris
kirk
spock
mccoy
krusher
janeway
picard
# Sparc 10 machines running SunOS
michael
jermaine
marlon
tito
jackie
```

Turns Engine Output Files

The naming convention for the NCD file, which may contain placement and routing information in varying degrees of completion, is placer_level_router_level_cost_table.ncd. If any of these elements are not used, they are represented by an *x*. For example, for the first design file run with the options –n 5 –t 16 –rl std –pl high, the NCD output file name would be high_std_16.ncd. The second file would be named high_std_17.ncd. For the first design file being run with the options –n 5 –t 16 –r –pl high, the NCD output file name would be high_x_16.ncd. The second file would be named high_x_17.ncd.

Limitations

The following limitations apply to the Turns Engine.

- The Turns Engine can operate only on Xilinx FPGA families. It cannot operate on CPLDs.
- Each run targets the same part, and uses the same algorithms and options. Only the starting point, or the cost table entry, is varied.

System Requirements

To use the Turns Engine, all of the nodes named in the nodelist must be able to access a common directory, most likely through a network-accessible file system.

If needed, one of the files in this directory can be used to set any environment variables that are needed to run PAR (e.g., XILINX, PATH, LD_LIBRARY_PATH). This can be accomplished as follows:

Create a file with a fixed path that can be accessed by all of the systems you are using. This file might be named something like:

/net/${nodename}/home/jim/parmsetup

In this file, add whatever lines are needed to set up environment variables that are needed to run PAR. This file will be interpreted by /bin/sh (Bourne shell) on each target node before starting PAR, so environment variables must be set using Bourne shell conventions. When running in a Solaris environment, the contents of this file might look like:

XILINX=/net/${nodename}/home/jim/xilinx

export XILINX

PATH=$XILINX/bin/sol:/usr/bin:/usr/sbin

export PATH

LD_LIBRARY_PATH=$XILINX/bin/sol

export LD_LIBRARY_PATH

For environments that mix different kinds of nodes (e.g., Solaris and Linux), a more sophisticated script might be needed to set up the proper environment.

After creating this file, set the environment variable PAR_M_SETUPFILE to the name of your file, as shown in the following examples.

Example for C shell:

setenv PAR_M_SETUPFILE /net/${nodename}/home/jim/parmsetup

Example for Bourne shell:

PAR_M_SETUPFILE=/net/${nodename}/home/jim/parmsetup

export PAR_M_SETUPFILE

Turns Engine Environment Variables

The following environment variables are interpreted by the Turns Engine manager.

- PAR_AUTOMNTPT—Specifies the network automount point. The Turns Engine uses network path names to access files. For example, a local path name to a file may be designs/cpu.ncd, but the network path name may be /home/*machine_name*/ivan/designs/cpu.ncd or /net/*machine_name*/ivan/designs/cpu.ncd. The PAR_AUTOMNT environment variable should be set to the value of the network automount point. The automount points for the examples above are /home and /net. The default value for PAR_AUTOMNT is /net.

  The following command sets the automount point to /nfs. If the current working directory is /usr/*user_name*/*design_name* on node mynode, the command **cd /nfs/mynode/usr/***user_name*/*design_name* is generated before PAR runs on the machine.

  ```
  setenv PAR_AUTOMNTPT /nfs
  ```

  The following setting does not issue a **cd** command; you are required to enter full paths for all of the input and output file names.

  ```
  setenv PAR_AUTOMNTPT ""
  ```

  The following command tells the system that paths on the local workstation are the same as paths on remote workstations. This can be the case if your network does not use an automounter and all of the mounts are standardized, or if you do use an automounter and all mount points are handled generically.

  ```
  setenv PAR_AUTOMNTPT "/"
  ```

- PAR_AUTOMNTTMPPT—Most networks use the /tmp_mnt temporary mount point. If your network uses a temporary mount point with a different name, like /t_mnt, then you must set the PAR_AUTOMNTTMPPT variable to the temporary mount point name. In the example above you would set PAR_AUTOMNTTMPPT to /t_mnt. The default value for PAR_AUTOMNTTMPPT is /tmp_mnt.

- PAR_M_DEBUG—Causes the Turns Engine to run in debug mode. If the Turns Engine is causing errors that are difficult to correct, you can run PAR in debug mode as follows:
  - Set the PAR_M_DEBUG variable:

    ```
    setenv PAR_M_DEBUG 1
    ```

  - Create a node list file containing only a single entry (one node). This single entry is necessary because if the node list contains multiple entries, the debug information from all of the nodes is intermixed, and troubleshooting is difficult.
  - Run PAR with the –m (multi-tasking mode) option. In debug mode, all of the output from all commands generated by the PAR run is echoed to the screen. There are also additional checks performed in debug mode, and additional information supplied to aid in solving the problem.
- PAR_M_SETUPFILE—See "System Requirements" for a discussion of this variable.

Debugging

With the Turns Engine you may receive messages from the login process. The problems are usually related to the network or to environment variables.

- Network Problem—You may not be able to logon to the machines listed in the nodelist file.
  - Use the following command to contact the nodes:

    ping *machine_name*

    You should get a message that the machine is running. The ping command should also be in your path (UNIX cmd: which ping).
  - Try to logon to the nodes using the command rsh *machine_ name*. You should be able to logon to the machine. If you cannot, make sure rsh is in your path (UNIX cmd: which rsh). If rsh is in your path, but you still cannot logon, contact your network administrator.
  - Try to launch PAR on a node by entering the following command.

    rsh *machine_name* **/bin/sh** –**c par**.

    This is the same command that the Turns Engine uses to launch PAR. If this command is successful, everything is set up correctly for the *machine_name* node.
- Environment Problem—logon to the node with the problem by entering the following UNIX command:

  rsh *machine_name*

  Check the `$XILINX`, `$LD_LIBRARY_PATH`, and `$PATH` variables by entering the UNIX command **echo $** *variable_name*. If these variables are not set correctly, check to make sure these variables are defined in your .cshrc file.

  Note: Some, but not all, errors in reading the .cshrc may prevent the rest of the file from being read. These errors may need to be corrected before the XILINX environment variables in the .cshrc are read. The error message /bin/sh: par not found indicates that the environment in the .cshrc file is not being correctly read by the node.

Screen Output

When PAR is running multiple jobs and is not in multi-tasking mode, output from PAR is displayed on the screen as the jobs run. When PAR is running multiple jobs in multi-tasking mode, you only see information regarding the current status of the Turns Engine. For example, when the job described in "Turns Engine Overview" is executed, the following screen output would be generated.

```
Starting job high_high_1 on node jupiter
Starting job high_high_2 on node mars
Starting job high_high_3 on node mercury
Starting job high_high_4 on node neptune
Starting job high_high_5 on node pluto
```

When one of the jobs finishes, a message similar to the following is displayed.

```
Finished job high_high_3 on node mercury
```

These messages continue until there are no jobs left to run, at which time *Finished* appears on your screen.

Note: For HP workstations, you are not able to interrupt the job with Ctrl+C as described following if you do not have Ctrl+C set as the escape character. To set the escape character, refer to your HP manual.

You may interrupt the job at any time by pressing Ctrl+C. If you interrupt the program, the following options are displayed:

1. **Continue processing and ignore the interrupt**—self-explanatory.
2. **Normal program exit at next check point**—allows the Turns Engine to wait for all jobs to finish before terminating. PAR is allowed to generate the master PAR output file (PAR), which describes the overall run results

   When you select option 2, a secondary menu appears as follows:

   a. **Allow jobs to finish** — current jobs finish but no other jobs start if there are any. For example, if you are running 100 jobs (–n 100) and the current jobs running are high_high_49 and high_high_50, when these jobs finish, job high_high_51 is not started.

   b. **Halt jobs at next checkpoint** — all current jobs stop at the next checkpoint; no new jobs are started.

   c. **Halt jobs immediately** — all current jobs stop immediately; no other jobs start
3. **Exit program immediately** — all running jobs stop immediately (without waiting for running jobs to terminate) and PAR exits the Turns Engine.
4. **Add a node for running jobs** — allows you to dynamically add a node on which you can run jobs. When you make this selection, you are prompted to input the name of the node to be added to the list. After you enter the node name, a job starts immediately on that node and a *Starting job* message is displayed.

5. **Stop using a node** — allows you to remove a node from the list so that no job runs on that node.

   If you select **Stop using a node**, you must also select from the following options.

   Which node do you wish to stop using?
   1. jupiter
   2. mars
   3. mercury

   Enter number identifying the node.(<CR> to ignore)

   Enter the number identifying the node. If you enter a legal number, you are asked to make a selection from this menu.

   Do you wish to
   1. Terminate the current job immediately and resubmit.
   2. Allow the job to finish.

   Enter number identifying choice. (<CR> to ignore)

   The options are described as follows:

   a. **Terminate the current job immediately and resubmit**—halts the job immediately and sets it up again to be run on the next available node. The halted node is not used again unless it is enabled by the *add* function.

   b. **Allow the job to finish**—finishes the node's current job, then disables the node from running additional jobs.

      Note: The list of nodes described above is not necessarily numbered in a linear fashion. Nodes that are disabled are not displayed. For example, if NODE2 is disabled, the next time *Stop using a node* is opted, the following is displayed.

      Which node do you wish to stop using?
      1. jupiter
      3. mercury

      Enter number identifying the node. (<CR> to ignore)

6. **Display current status** — displays the current status of the Turns Engine. It shows the state of nodes and the respective jobs. Here is a sample of what you would see if you chose this option.

ID	NODE	STATUS	JOB	TIME
1.	jupiter	Job Running	high_high_10	02:30:45
2.	mars	Job Running	high_high_11	02:28:03
3.	mercury	Not Available		
4.	neptune	Pending Term	high_high_12	02:20:01

A few of the entries are described as follows:

- jupiter has been running job high_high_10 for approximately 2 1/2 hours.
- mars has been running job high_high_11 for approximately 2 1/2 hours.
- mercury has been deactivated by the user with the *Stop using a node* option or it was not an existing node or it was not running. Nodes are *pinged* to see if they exist and are running before attempting to start a job.
- neptune has been halted *immediately* with job resubmission. The Turns Engine is waiting for the job to terminate. Once this happens the status is changed to *not available*.

There is also a *Job Finishing* status. This appears if the Turns Engine has been instructed to halt the job at the next checkpoint.

Halting PAR

You need to set the interrupt character by entering **stty intr ^V^C** in the .login file or .cshrc file.

Note: You cannot halt PAR with Ctrl+C if you do not have Ctrl+C set as the interrupt character. To halt a PAR operation, enter Ctrl+C. In a few seconds, the following message appears:

```
CNTRL-C interrupt detected.

Please choose one of the following options:
1. Ignore interrupt and continue processing.
2. Exit program normally at next checkpoint. This saves the best results
   so far after concluding the current processing,
3. Exit program immediately.
4. Display Failing Timespec Summary.
5. Cancel the current job and move to the next one at the next check
   point.
Enter choice -->
```

If you have no failing time specifications or are not using the –n option, Options 4 and 5 display as follows.

```
4. Display Failing Timespec Summary.
   (Not applicable: Data not available)
5. Cancel the current job and move to the next one at
   the next check point.
    (Not applicable: Not a multi-run job.)
```

You then select one of the five options shown on the screen. The description of the options are as follows:

- Option 1—this option causes PAR to continue operating as before the interruption. PAR then runs to completion.
- Option 2—this option continues the current place/route iteration until one of the following *check points*.
  - After placement
  - After the current routing phase

  The system then exits the PAR run and saves an intermediate output file containing the results up to the check point.

If you use this option, you may continue the PAR operation at a later time. To do this, you must look in the PAR report file to find the point at which you interrupted the PAR run. You can then run PAR on the output NCD file produced by the interrupted run, setting command line options to continue the run from the point at which it was interrupted.

Option 2 halt during routing may be helpful if you notice that the router is performing multiple passes without improvement, and it is obvious that the router is not going to achieve 100% completion. In this case, you may want to halt the operation before it ends and use the results to that point instead of waiting for PAR to end by itself.

- Option 3—this option stops the PAR run immediately. You do not get any output file for the current place/route iteration. You do, however, still have output files for previously completed place/route iterations.
- Option 4—this option is currently disabled.
- Option 5—Terminates current iteration if you have used the –n option and continues the next iteration.

Note: If you started the PAR operation as a background process on a workstation, you must bring the process to the foreground using the **`fg`** command before you can halt the PAR operation.

After you run PAR, you can use the FPGA Editor on the NCD file to examine and edit the results. You can also perform a static timing analysis using TRACE or the Timing Analyzer. When the design is routed to your satisfaction, you can input the resulting NCD file into the Xilinx Development System's BitGen program. BitGen creates files that are used for downloading the design configuration to the target FPGA. For details on BitGen, see Chapter 14, "BitGen".

Chapter 10

XPower

This program is compatible with the following device families:

- Virtex™, Virtex™-E
- Virtex™-II
- Virtex™-II Pro, Virtex™-II Pro X
- Virtex™-4
- Virtex™-5 LX
- Spartan™-II, Spartan™-IIE
- Spartan™-3, Spartan™-3E, Spartan™-3L
- CoolRunner™ XPLA3, CoolRunner™-II
- XC9500™, XC9500XL™, XC9500XV™

For a complete list of supported devices:

Run **`xpwr -ls [-arch]`** from the command line. See "-ls (List Supported Devices)" for additional information on this option. You can also view a list of supported devices and get more XPower information from the online help included in the XPower GUI tool.

This chapter contains the following sections:

XPower Overview

XPower provides power and thermal estimates after PAR, for FPGA designs, and after CPLDfit, for CPLD designs. XPower does the following:

- Estimates how much power the design will consume
- Identifies how much power each net or logic element in the design is consuming
- Verifies that junction temperature limits are not exceeded

Files Used by XPower

XPower uses the following file types:

- CXT - File produced by CPLDfit and used by XPower to calculate and display power consumption.
- NCD - An physical design file produced by MAP and PAR that contains information on an FPGA. Xilinx recommends using a fully placed and routed NCD design (produced by PAR) to get the most accurate power estimate. Using a mapped-only NCD (produced by MAP) file may compromise accuracy.
- PCF - An optional user-modifiable ASCII Physical Constraints File (PCF) produced by MAP. The PCF contains timing constraints that XPower uses to identify clock nets (by using the period constraint) and GSRs (by looking at TIGs). Temperature and voltage information is also available if these constraints have been set in the User Constraints FIle (UCF).

  Note: The Innoveda CAE tools create a file with a .pcf extension when generating a plot of an Innoveda schematic. This PCF is not related to a Xilinx PCF. Because XPower automatically reads a PCF with the same root name as your design file, make sure your directory does not contain an Innoveda PCF with the same root name as your NCD file.
- VCD - Output file from simulators. XPower uses this file to set frequencies and activity rates of internal signals, which are signals that are not inputs or outputs but internal to the design. For a list of supported simulators, see the "VCD Data Entry" section of this chapter.
- XAD - Output file that provides most of the information contained in the VCD file, but is much smaller in size to reduce runtime.
- XML - Settings file from XPower. Settings for a design can be saved to an XML file and then reloaded into XPower for the same design. Data input such as frequencies, toggle rates, and capacitance loads can be saved to this file to avoid entering the same information the next time the design is loaded into XPower.

XPower Syntax

Use the following syntax to run XPower from the command line:

FPGA Designs

```
xpwr infile[.ncd] [constraints_file[.pcf]] [options] -o design_name.pwr
```

CPLD Designs

```
xpwr infile[.cxt] [options] -o design_name.pwr
```

infile is the name of the input physical design file. If you enter a filename with no extension, XPower looks for an NCD file with the specified name. If no NCD file is found, XPower looks for a CXT file.

constraints file is the name of the Physical Constraints File (PCF). This optional file is used to define timing constraints for the design. If you do not specify a PCF, XPower looks for one with the same root name as the input NCD file. If a CXT file is found, XPower does not look for a PCF file.

options is one or more of the XPower options listed in "Command Line Options". Separate multiple options with spaces.

design name is the name of the output power report file with a .pwr extension. If a file name is not specified with the -o option, by default XPower generates a .pwr file with the same root name as the *infile*.

Using XPower

This section describes the settings necessary to obtain accurate power and thermal estimates, and the methods that XPower allows. This section refers specifically to FPGA designs. For CPLD designs, please see Xilinx Application Note XAPP360 at http://www.xilinx.com/support.

VCD Data Entry

The recommended XPower flow uses a VCD file generated from post PAR simulation. To generate a VCD file, you must have a Xilinx supported simulator. See the *Synthesis and Simulation Design Guide* for more information.

XPower supports the following simulators:

- ISIM
- ModelSim
- Cadence Verilog XL
- Cadence NC-Verilog
- Cadence NC-VHDL
- Cadence NC-SIM
- Synopsys VCS
- Synopsys Scirocco

XPower uses the VCD file to set toggle rates and frequencies of all the signals in the design. Manually set the following:

- Voltage (if different from the recommended databook values)
- Ambient temperature (default = 25 degrees C)
- Output loading (capacitance and current due to resistive elements)

For the first XPower run, voltage and ambient temperature can be applied from the PCF, provided temperature and voltage constraints have been set.

Xilinx recommends creating a settings file (XML). A settings file saves time if the design is reloaded into XPower. All settings (voltage, temperature, frequencies, and output loading) are stored in the settings file. See the "-wx (Write XML File)" section of this chapter for more information.

Other Methods of Data Entry

All asynchronous signals are set using an absolute frequency in MHz. All synchronous signals are set using activity rates.

An activity rate is a percentage between 0 and 100. It refers to how often the output of a registered element changes with respect to the active edges of the clock. For example, a 100MHz clock going to a flip flop with a 100% activity rate has an output frequency of 50MHz.

When using other methods of design entry, you must set the following:

- Voltage (if different from the recommended databook values)
- Ambient temperature (default = 25 degrees C)
- Output loading (capacitance and current due to resistive elements)
- Frequency of all input signals
- Activity rates for all synchronous signals

If you do not set activity rates, XPower assumes 0% for all synchronous nets. The frequency of input signals is assumed to be 0MHz. The default ambient temperature is 25 degrees C. The default voltage is the recommended operating voltage for the device.

Note: The accuracy of the power and thermal estimates is compromised if you do not set all of the above mentioned signals. At a minimum, you should set high power consuming nets, such as clock nets, clock enables, and other fast or heavily loaded signals and output nets.

Command Line Options

This section describes the XPower command line options. For a list of available options from the command line, run **`xpwr -h`**.

-l (Limit)

```
-l [limit]
```

Imposes a line limit on the verbose report. An integer value must be specified as an argument. The integer represents the output number of lines in the report file.

-ls (List Supported Devices)

```
-ls [architecture]
```

Lists the supported Xilinx devices in the current software installation. Use the *architecture* argument to specify a specific architecture, for example Virtex.

-o (Rename Power Report)

```
-o reportName.pwr
```

Specifies the name of the output power report file. If a filename is not specified, the output power report is the input design filename with a .pwr extension.

-s (Specify VCD file)

```
-s [simdata.vcd]
```

Sets activity rates and signal frequencies using data from the VCD file. If no file is specified, XPower searches for an input design file with a .vcd extension.

-tb (Turn On Time Based Reporting)

```
-tb [number][unit]
```

Turns on time-based reporting. Must be used with the -s option. XPower generates a file with a .txt extension. The specified *number* and *unit* control the time base of how often the total power is reported. For example, if 10ps is selected, XPower reports the total instantaneous power every 10 picoseconds of the simulation run. If the simulation runtime for the VCD is 100ps, XPower returns 10 results.

Units are:

ps = picoseconds

ns = nanoseconds

fs = femtoseconds

us = microseconds

-v (Verbose Report)

```
-v [-a]
```

Specifies a verbose (detailed) power report. Use of the -a argument specifies an advanced report. See "Power Reports" below for more information.

-wx (Write XML File)

```
-wx [userdata.xml]
```

Writes out an XML settings file that contains all of the settings information from the current XPower run. If no filename is specified, the output filename is the input design filename with a .xml extension.

-x (Specify Settings (XML) Input File)

```
-x [userdata.xml]
```

Instructs XPower to use an existing XML settings file to set the frequencies of signals and other values. If an XML file is not specified, XPower searches for a file with the input design filename and a .xml extension.

Command Line Examples

The following command produces a standard report, **mydesign.pwr**, in which the VCD file specifies the activity rates and frequencies of signals. The output loading has not been changed; all outputs assume the default loading of 10pF. The design is for FPGAs.

```
xpwr mydesign.ncd mydesign.pcf -s timesim.vcd
```

The following command does all of the above and generates a settings file called **mysettings.xml**. The settings file contains all of the information from the VCD file.

```
xpwr mydesign.ncd mydesign.pcf -s timesim.vcd -wx mysettings.xml
```

The following command does all of the above and generates a detailed (verbose) report instead of a standard report. The verbose report is limited to 100 lines.

```
xpwr mydesign.ncd mydesign.pcf -v -l 100 -s timesim.vcd -wx
mysettings.xml
```

Power Reports

This section explains what you can expect to see in a power report. Power reports have a .pwr extension.

There are three types of power reports:

- "Standard Reports" (the default)
- "Detailed Reports" (the report generated when you run the "-v (Verbose Report)" command line option)
- "Advanced Reports"

Standard Reports

A standard report contains the following:

- A report header specifying:
  - The XPower version
  - A copyright message
  - Information about the design and associated files, including the design filename and any PCF and simulation files loaded
  - The data version of the information
- The Power Summary, which gives the power and current totals as well as other summary information.
- The Thermal Summary, which consists of:
  - Airflow
  - Estimated junction temperature
  - Ambient temperature
  - Case temperature
  - Theta J-A

- A Decoupling Network Summary, which contains capacitance values, recommendations, and a total for each voltage source broken down in individual capacitance ranges.
- A footer containing the analysis completion date and time.

Detailed Reports

A detailed power report includes all of the information in a standard power report, plus power details listed for logic, signals, clocks, inputs, and outputs of the design.

Advanced Reports

An advanced report includes all the information in a standard report, plus the following information:

- The maximum power that can be dissipated under the specified package, ambient temperature, and cooling conditions
- Heatsink and glue combination
- An upper limit on the junction temperature that the device can withstand without breaching recommended limits
- Power details, including individual elements by type
- I/O bank details for the decoupling network
- Element name, the number of loads, the capacitive loading, the capacitance of the item, the frequency, the power, and the current

Note: The number of loads is reported only for signals. The capacitive loading is reported only for outputs. If the capacitance is zero, and there is a non-zero frequency on an item, the power is shown to be "~0", which represents a negligible amount of power.

Chapter 11

PIN2UCF

This program is compatible with the following families:

- Virtex™, Virtex™-E
- Virtex™-II
- Virtex™-II Pro, Virtex™-II Pro X
- Virtex™-4
- Virtex™-5 LX
- Spartan™-II, Spartan™-IIE
- Spartan™-3, Spartan™-3E, Spartan™-3L
- CoolRunner™ XPLA3, CoolRunner™-II
- XC9500™, XC9500XL™, XC9500XV™

This chapter describes the PIN2UCF program. The chapter contains the following sections:

PIN2UCF Overview

PIN2UCF is a command line program that generates pin-locking constraints in a User Constraints File (UCF) by reading a placed, or placed and routed NCD file for FPGAs or GYD file for CPLDs. PIN2UCF writes its output to an existing UCF. If there is no existing UCF, PIN2UCF creates one.

The following figure shows the flow through PIN2UCF.

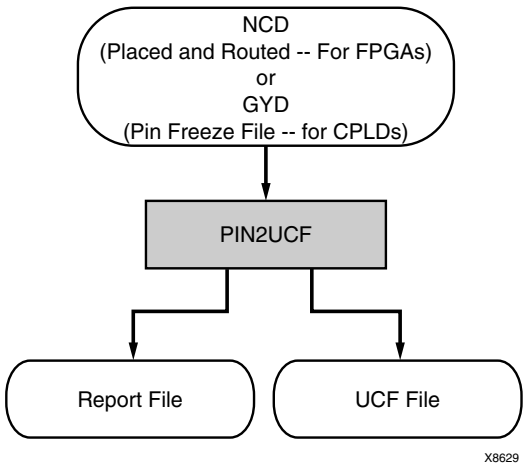


Figure 11-1: **PIN2UCF Flow**

The PIN2UCF program is used to back-annotate pin-locking constraints to the UCF from a successfully placed and routed design for an FPGA or a successfully fitted design for a CPLD.

PIN2UCF extracts pin locations and logical pad names from an existing NCD or GYD file and writes this information to a UCF. Pin-locking constraints are written to a PINLOCK section in the UCF. The PINLOCK section begins with the statement #PINLOCK BEGIN and ends with the statement #PINLOCK END. By default, PIN2UCF does not write conflicting constraints to a UCF. Prior to creating a PINLOCK section, if PIN2UCF discovers conflicting constraints, it writes information to a report file named pinlock.rpt.

The pinlock.rpt file has the following sections:

- Constraints Conflicts Information

  This section has the following subsections. If there are no conflicting constraints, both subsections contain a single line indicating that there are no conflicts.

  - Net name conflicts on the pins
  - Pin name conflicts on the nets

  Note: This section does not appear if there are fatal input errors, for example, missing inputs or invalid inputs.

- List of Errors and Warnings

  This section appears only if there are errors or warnings.

User-specified pin-locking constraints are never overwritten in a UCF. However, if the user-specified constraints are exact matches of PIN2UCF generated constraints, a pound sign (#) is added in front of all matching user-specified location constraint statements. The pound sign indicates that a statement is a comment. To restore the original UCF (the file without the PINLOCK section), remove the PINLOCK section and delete the pound sign from each of the user-specified statements.

Note: PIN2UCF does not check if existing constraints in the UCF are valid pin-locking constraints.

PIN2UCF writes to an existing UCF under the following conditions:

- The contents in the PINLOCK section are all pin lock matches, and there are no conflicts between the PINLOCK section and the rest of the UCF.
- The PINLOCK section contents are all comments and there are no conflicts outside of the PINLOCK section.
- There is no PINLOCK section and no other conflicts in the UCF.

Note: Comments inside an existing PINLOCK section are never preserved by a new run of PIN2UCF. If PIN2UCF finds a CSTTRANS comment, it equates "INST *name*" to "NET name" and then checks for comments.

PIN2UCF Syntax

The following command runs the PIN2UCF command line program:

pin2ucf {*ncd_file*.**ncd**|*pin_freeze_file*.**gyd**} [**–r** *report_file_name* **-o** *output*.**ucf**]

ncd_file or *pin_freeze_file* is the name of the input placed and routed NCD file for FPGAs, or the fitted GYD file for CPLDs.

PIN2UCF Input Files

PIN2UCF uses the following files as input:

- NCD file—The minimal requirement is a placed NCD file, but you would normally use a placed and routed NCD file that meets (or is fairly close to meeting) timing specifications.
- GYD file—The PIN2UCF pin-locking utility replaces the old GYD file mechanism that was used by CPLDs to lock pins. The GYD file is still available as an input guide file to control pin-locking. Running PIN2UCF is the recommended method of pin-locking to be used instead of specifying the GYD file as a guide file.

PIN2UCF Output Files

PIN2UCF creates the following files as output:

- UCF—If there is no existing UCF, PIN2UCF creates one. If an *output*.ucf file is not specified for PIN2UCF and a UCF with the same root name as the design exists in the same directory as the design file, the program writes to that file automatically unless there are constraint conflicts.
- RPT file— A pinlock.rpt file is written to the current directory by default. Use the –r option to write a report file to another directory. See "–r (Write to a Report File)" for more information.

PIN2UCF Options

This section describes the PIN2UCF command line options.

–o (Output File Name)

–o *outfile*.**ucf**

By default (without the –o option), PIN2UCF writes an *ncd_file*.ucf file. The –o option specifies the name of the output UCF for the design. You can use the –o option if the UCF used for the design has a different root name than the design name. You can also use this option to write the pin-locking constraints to a UCF with a different root name than the design name, or if you want to write the UCF to a different directory.

–r (Write to a Report File)

–r *report_file_name*.rpt

By default (without the –r option), a pinlock.rpt file is automatically written to the current directory. The –r option writes the PIN2UCF report into the specified report file.

PIN2UCF Scenarios

The following table lists scenarios for the PIN2UCF program.

Table 11-1: **PIN2UCF Scenarios**

Scenario	PIN2UCF Behavior	Files Created or Updated
No UCF is present.	PIN2UCF creates a UCF and writes the pin-locking constraints to the UCF.	pinlock.rpt *design_name*.ucf
UCF is present. There are no pin-locking constraints in the UCF, or this file contains some user-specified pin-locking constraints outside of the PINLOCK section. None of the user-specified constraints conflict with the PIN2UCF generated constraints.	PIN2UCF appends the pin-locking constraints in the PINLOCK section to the end of the file.	pinlock.rpt *design_name*.ucf

Table 11-1: **PIN2UCF Scenarios**

Scenario	PIN2UCF Behavior	Files Created or Updated
UCF is present. The UCF contains some user-specified pin-locking constraints either inside or outside of the PINLOCK section. Some of the user-specified constraints conflict with the PIN2UCF generated constraints	PIN2UCF does not write the PINLOCK section. Instead, it exits after providing an error message. It writes a list of conflicting constraints.	pinlock.rpt
UCF is present. There are no pin-locking constraints in the UCF. There is a PINLOCK section in the UCF generated from a previous run of PIN2UCF or manually created by the user. None of the constraints in the PINLOCK section conflict with PIN2UCF generated constraints.	PIN2UCF writes a new PINLOCK section in the UCF after deleting the existing PINLOCK section. The contents of the existing PINLOCK section are moved to the new PINLOCK section.	pinlock.rpt *design_name*.ucf

Chapter 12

TRACE

The Timing Reporter and Circuit Evaluator (TRACE) program is compatible with the following device families:

- Virtex™, Virtex™-E
- Virtex™-II
- Virtex™-II Pro/X
- Virtex™-4
- Virtex™-5 LX
- Spartan™, Spartan™-II, Spartan™-IIE
- Spartan™-3, Spartan™-3E, Spartan™-3L
- CoolRunner™ XPLA3, CoolRunner™-II
- XC9500™, XC9500XL™, XC9500XV™

This chapter contains the following sections:

TRACE Overview

TRACE provides static timing analysis of an FPGA design based on input timing constraints.

TRACE performs two major functions.

- Timing Verification—Verifies that the design meets timing constraints.
- Reporting—Generates a report file that lists compliance of the design against the input constraints. TRACE can be run on unplaced designs, only placed designs, partially placed and routed designs, and completely placed and routed designs.

The following figure shows the primary inputs and outputs to TRACE. The NCD file is the output design file from MAP or PAR, which has a .ncd extension. The optional PCF is the physical constraints file, which has a .pcf extension. The TWR file is the timing report file, which has a .twr extension.

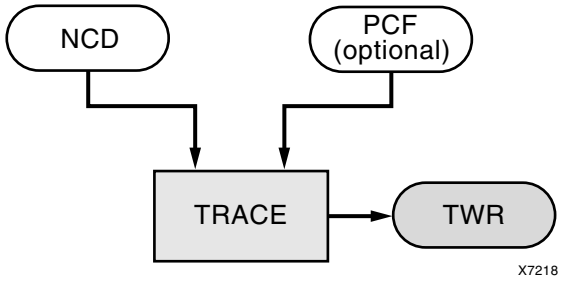


Figure 12-1: **TRACE flow with primary input and output files**

TRACE Syntax

Use the following syntax to run TRACE from the command line:

trce [*options*] *design*[**.ncd**] [*constraint*[**.pcf**]]

TRACE can also be used in conjunction with a macro file (XTM), where all traditional inputs (NCD and PCF) and outputs (timing reports) are optional, and the macro file is mandatory. The following syntax runs TRACE with the macro file option:

trce –run *macro***[.xtm]** *design*[**.ncd**] [*constraint*[**.pcf**]

Note: See the "–run (Run Timing Analyzer Macro)" section for more information.

constraint specifies the name of a physical constraints file (PCF). This file is used to define timing constraints for the design. If you do not specify a physical constraints file, TRACE looks for one with the same root name as the input design (NCD) file.

design specifies the name of the input design file. If you enter a file name with no extension, TRACE looks for an NCD file with the specified name.

macro specifies the name of the Timing Analyzer macro file (XTM). This file is used to produce timing reports based on the commands specified in the XTM file.

options can be any number of the command line options listed in the "TRACE Options" section of this chapter. Options need not be listed in any particular order unless you are using the –stamp (Generates STAMP timing model files) option. Separate multiple options with spaces.

TRACE Input Files

Input to TRACE can be a mapped, a placed, or a placed and routed NCD file, along with an optional physical constraints file (PCF). The PCF is produced by the MAP program and based on timing constraints that you specify. Constraints can show such things as clock speed for input signals, the external timing relationship between two or more signals, absolute maximum delay on a design path, and general timing requirements for a class of pins.

Input files to TRACE:

- NCD file—A mapped, a placed, or a placed and routed design. The type of timing information TRACE provides depends on whether the design is unplaced (after MAP), placed only, or placed and routed.
- PCF—An optional, user-modifiable, physical constraints file produced by MAP. The PCF contains timing constraints used when TRACE performs a static timing analysis.
- XTM file—A macro file, produced by Timing Analyzer, that contains a series of commands for generating custom timing reports with TRACE. See the *Timing Analyzer online help* for information on creating XTM files.

  Note: The Innoveda CAE tools create a file with a .pcf extension when they generate an Innoveda schematic. This PCF is not related to a Xilinx PCF. Because TRACE automatically reads a PCF with the same root name as your design file, make sure your directory does not contain an Innoveda PCF with the same root name as your NCD file.

TRACE Output Files

TRACE outputs the following timing reports based on options specified on the command line:

TWR—default timing report. The –e (error report) and –v (verbose report) options can be used to specify the type of timing report you want to produce: summary report (default), error report, or verbose report.

TWX—XML timing report output by using the –xml option. This report is viewable with the Timing Analyzer GUI tool. The –e (error report) and –v (verbose report) options apply to the TWX file as well as the TWR file. See the "–xml (XML Output File Name)" section for details.

LOG—log file created when the –run option is used. This .log file has the same root name as the input *macro*.xtm file.

TRACE generates an optional STAMP timing model with the –stamp option.
See the "–stamp (Generates STAMP timing model files)" section in this chapter for details.

Note: For more information on the types of timing reports that TRACE generates, see the "TRACE Reports" section in this chapter.

TRACE Options

This section describes the TRACE command line options.

–a (Advanced Analysis)

The –a option is only used if you are not supplying any timing constraints (from a PCF) to TRACE. The –a option writes out a timing report with the following information:

- An analysis that enumerates all clocks and the required OFFSETs for each clock.
- An analysis of paths having only combinatorial logic, ordered by delay.

This information is supplied in place of the default information for the output timing report type (summary, error, or verbose).

Note: An analysis of the paths associated with a particular clock signal includes a hold violation (race condition) check only for paths whose start and endpoints are registered on the same clock edge.

–e (Generate an Error Report)

–e [*limit*]

The –e option causes the timing report to be an error report instead of the default summary report. See "Error Report" for a sample error report.

The report has the same root name as the input design and has a .twr extension.

The optional *limit* is an integer limit on the number of items reported for each timing constraint in the report file. The value of *limit* must be an integer from 0 to 32,000 inclusive. The default is 3.

–f (Execute Commands File)

-f *command_file*

The –f option specifies the command file to use as input.

–fastpaths (Report Fastest Paths)

-fastpaths

The –fastpaths option is used to report the fastest paths of a design.

Note: This option is being deprecated in this release and will not be available in future releases of Xilinx software.

–intstyle (Integration Style)

-intstyle {**ise** | **xflow** | **silent**}

The –intstyle option reduces screen output based on the integration style you are running. When using the –intstyle option, one of three modes must be specified: *ise*, *xflow*, or *silent*. The mode sets the way information is displayed in the following ways:

-intstyle ise

This mode indicates the program is being run as part of an integrated design environment.

-intstyle xflow

This mode indicates the program is being run as part of an integrated batch flow.

-intstyle silent

This mode limits screen output to warning and error messages only.

Note: The -intstyle option is automatically invoked when running in an integrated environment, such as Project Navigator or XFLOW.

–ise (ISE Project File)

-ise *project_file*

The –ise option specifies an ISE project file, which can contain settings to capture and filter messages produced by the program during execution.

–l (Limit Timing Report)

-l *limit*

The –l option limits the number of items reported for each timing constraint in the report file. The limit value must be an integer from 0 to 2,000,000,000 (2 billion) inclusive. If a limit is not specified, the default value is 3.

Note: The higher the limit value, the longer it takes to generate the timing report.

–nodatasheet (No Data Sheet)

-nodatasheet

The –nodatasheet option does not include the datasheet section of a generated report.

–o (Output Timing Report File Name)

–o *report*[**.twr**]

The –o option specifies the name of the output timing report. The .twr extension is optional. If –o is not used, the output timing report has the same root name as the input design (NCD) file.

–run (Run Timing Analyzer Macro)

–run *macro*.**xtm** [*design*.**ncd**] [*constraints*.**pcf**]

Commands used to generate custom timing reports with Timing Analyzer can be saved in a macro file and run in batch mode with the TRACE program. The –run option uses a Timing Analyzer macro file (XTM) to produce timing reports based on the commands specified in the XTM file. The –run option also generates a log file (*macro_file*.log) that lists all of the Timing Analyzer macro commands and any status messages.

Optional design and PCF files can be specified on the command line and will be opened before TRACE runs the XTM macro file. If a design is not specified on the command line, the XTM file must include the macro command to open a design. If an output report filename and type are not specified in the macro file, TRACE generates an XML timing report (TWX) file by default and names the report Timing1.twx. When multiple timing reports and generated by the macro file, TRACE uses the same naming convention: Timing1.twx, Timing2.twx, Timing3.twx, and so on.

Note: For information on creating an XTM macro file, see the Timing Analyzer online help.

–s (Change Speed)

–s [*speed*]

The –s option overrides the device speed contained in the input NCD file and instead performs an analysis for the device speed you specify. The –s option applies to whichever report type you produce in this TRACE run. The option allows you to see if faster or slower speed grades meet your timing requirements.

The device *speed* can be entered with or without the leading dash. For example, both **–s 3** and **–s –3** are valid entries.

Some architectures support minimum timing analysis. The command line syntax for min timing analysis is: trace –s min. Do not place a leading dash before min.

Note: The –s option only changes the speed grade for which the timing analysis is performed; it does not save the new speed grade to the NCD file.

–skew (Analyze Clock Skew for All Clocks)

–skew

This option is obsolete. By default, TRACE now analyzes clock skew and hold time violations for all clocks, including those using general clock routing resources.

–stamp (Generates STAMP timing model files)

–stamp *stampfile design*.**ncd**

When you specify the –stamp option, TRACE generates a pair of STAMP timing model files--stampfile.mod and stampfile.data--that characterize the timing of a design.

Note: The stamp file entry must precede the NCD file entry on the command line.

The STAMP compiler can be used for any printed circuit board when performing static timing analysis.

Methods of running TRACE with the STAMP option to obtain a complete STAMP model report are as follows:

- Run with advanced analysis using the –a option.
- Run using default analysis (with no constraint file and without advanced analysis).
- Construct constraints to cover all paths in the design.
- Run using the unconstrained path report (–u option) for constraints which only partially cover the design.

For either of the last two options, do not include TIGs in the PCF, as this can cause paths to be excluded from the model.

–u (Report Uncovered Paths)

–u *limit*

The –u option reports delays for paths that are not covered by timing constraints. The option adds an *unconstrained path analysis* constraint to your existing constraints. This constraint performs a default path enumeration on any paths for which no other constraints apply. The default path enumeration includes circuit paths to data and clock pins on sequential components and data pins on primary outputs.

The optional limit argument limits the number of unconstrained paths reported for each timing constraint in the report file. The value of *limit* must be an integer from 1 to 2,000,000,000 (2 billion) inclusive. If a limit is not specified, the default value is 3.

In the TRACE report, the following information is included for the unconstrained path analysis constraint.

- The minimum period for all of the uncovered paths to sequential components.
- The maximum delay for all of the uncovered paths containing only combinatorial logic.
- For a verbose report only, a listing of periods for sequential paths and delays for combinatorial paths. The list is ordered by delay value in descending order, and the number of entries in the list can be controlled by specifying a limit when you enter the –v (Generate a Verbose Report) command line option.

Note: Register-to-register paths included in the unconstrained path report undergoes a hold violation (race condition) check only for paths whose start and endpoints are registered on the same clock edge.

–v (Generate a Verbose Report)

–v *limit*

The –v option generates a verbose report. The report has the same root name as the input design and a .twr. You can assign a different root name for the report on the command line, but the extension must be .twr.

The optional *limit* used to limit the number of items reported for each timing constraint in the report file. The value of *limit* must be an integer from 1 to 32,000 inclusive. If a limit is not specified, the default value is 3.

–xml (XML Output File Name)

The –xml option specifies the name of the output XML timing report (TWX) file. The .twx extension is optional.

–xml *outfile* [**.twx**]

Note: The XML report is not formatted and can only be viewed with the Timing Analyzer GUI tool. For more information on Timing Analyzer, see the online help provided with the tool.

TRACE Command Line Examples

The following command verifies the timing characteristics of the design named **design1.ncd**, generating a summary timing report. Timing constraints contained in the file **group1.pcf** are the timing constraints for the design. This generates the report file **design1.twr**.

```
trce design1.ncd group1.pcf
```

The following command verifies the characteristics for the design named **design1.ncd**, using the timing constraints contained in the file **group1.pcf** and generates a verbose timing report. The verbose report file is called **output.twr**.

```
trce –v 10 design1.ncd group1.pcf –o output.twr
```

The following command verifies the timing characteristics for the design named **design1.ncd**, using the timing constraints contained in the file **group1.pcf**, and generates a verbose timing report (TWR report and XML report). The verbose report file is called **design1.twr**, and the verbose XML report file is called **output.twx**.

```
trce –v 10 design1.ncd group1.pcf –xml output.twx
```

The following command verifies the timing characteristics for the design named **design1.ncd** using the timing constraints contained in the timing file .pcf, and generates an error report. The error report lists the three worst errors for each constraint in timing .pcf. The error report file is called **design1.twr**.

```
trce   -e 3   design1.ncd   timing.pcf
```

TRACE Reports

Default output from TRACE is an ASCII formatted timing report file that provides information on how well the timing constraints for the design are met. The file is written into your working directory and has a .twr extension. The default name for the file is the root name of the input NCD file. You can designate a different root name for the file, but it must have a .twr extension. The .twr extension is assumed if not specified.

The timing report lists statistics on the design, any detected timing errors, and a number of warning conditions.

Timing errors show absolute or relative timing constraint violations, and include the following:

- Path delay errors—where the path delay exceeds the MAXIMUM DELAY constraint for a path.
- Net delay errors—where a net connection delay exceeds the MAXIMUM DELAY constraint for the net.
- Offset errors—where either the delay offset between an external clock and its associated data-in pin is insufficient to meet the timing requirements of the internal logic or the delay offset between an external clock and its associated data-out pin exceeds the timing requirements of the external logic.
- Net skew errors—where skew between net connections exceeds the maximum skew constraint for the net.

To correct timing errors, you may need to modify your design, modify the constraints, or rerun PAR.

Warnings point out potential problems, such as circuit cycles or a constraint that does not apply to any paths.

Three types of reports are available: summary, error, and verbose. You determine the report type by entering the corresponding TRACE command line option, or by selecting the type of report when using the Timing Analyzer GUI tool (see "TRACE Options"). Each type of report is described in "Reporting with TRACE."

In addition to the ASCII formatted timing report (TWR) file, you can generate an XML timing report (TWX) file with the –xml option. The XML report is not formatted and can only be viewed with the Timing Analyzer GUI tool.

Note: The Constraints Interaction report (TSI), also referred to as the Timing Specification Interaction report, is no longer available.

Timing Verification with TRACE

TRACE checks the delays in the input NCD file against your timing constraints. If delays are exceeded, TRACE issues the appropriate timing error.

Note: Xilinx recommends limiting timing constraint values to 2 ms (milliseconds). Timing Constraint values more than 2 ms may result in bad values in the timing report.

Net Delay Constraints

When a MAXDELAY constraint is used, the delay for a constrained net is checked to ensure that the routedelay is less than or equal to the NETDELAY constraint.

routedelay ≤ **netdelayconstraint**

ROUTEDELAY is the signal delay between the driver pin and the load pins on a net. This is an estimated delay if the design is placed but not routed.

Any nets with delays that do not meet this condition generate timing errors in the timing report.

Net Skew Constraints

When using USELOWSKEWLINES or MAXSKEW constraints or "–skew (Analyze Clock Skew for All Clocks)", signal skew on a net with multiple load pins is the difference between minimum and maximum load delays.

```
signalskew = (maxdelay - mindelay)
```

MAXDELAY is the maximum delay between the driver pin and a load pin.

MINDELAY is the minimum delay between the driver pin and a load pin.

Note: Register-to-register paths included in a MAXDELAY constraint report undergo a hold violation (race condition) check only for paths whose start and endpoints are registered on the same clock edge.

For constrained nets in the PCF, skew is checked to ensure that the SIGNALSKEW is less than or equal to the MAXSKEW constraint.

signalskew ≤ **maxskewconstraint**

If the skew exceeds the maximum skew constraint, the timing report shows a skew error.

Path Delay Constraints

When a PERIOD constraint is used, the pathdelay equals the sum of logic (component) delay, route (wire) delay, and setup time (if any), minus clock skew (if any).

```
pathdelay = logicdelay + routedelay + setuptime - clockskew
```

The delay for constrained paths is checked to ensure that the pathdelay is less than or equal to the MAXPATHDELAY constraint.

pathdelay ≤ **maxpathdelayconstraint**

The following table lists the terminology for path delay constraints:

Table 12-1: **Path Delay Constraint Terminology**

Term	Definition
logicdelay	Pin-to-pin delay through a component.
routedelay	Signal delay between component pins in a path. This is an estimated delay if the design is placed but not routed.
setuptime	For clocked paths only, the time that data must be present on an input pin before the arrival of the triggering edge of a clock signal.
clockskew	For register-to-register clocked paths only, the difference between the amount of time the clock signal takes to reach the destination register and the amount of time the clock signal takes to reach the source register. Clock skew is discussed in the following section.

Paths showing delays that do not meet this condition generate timing errors in the timing report.

Clock Skew and Setup Checking

Clock skew must be accounted for in register-to-register setup checks. For register-to-register paths, the data delay must reach the destination register within a single clock period. The timing analysis software ensures that any clock skew between the source and destination registers is accounted for in this check.

Note: Clock skew must be accounted for in register-to-register setup checks. For register-to-register paths, the data delay must reach the destination register within a single clock period. The timing analysis software ensures that any clock skew between the source and destination registers is accounted for in this check. By default, the clock skew of all non-dedicated clocks, local clocks, and dedicated clocks is analyzed.

A setup check performed on register-to-register paths checks the following condition:

```
Slack = constraint + Tsk - (Tpath + Tsu)
```

The following table lists the terminology for clock skew and setup checking:

In the following figure, the clock skew Tsk is the delay from the clock input (CLKIOB) to

Table 12-2: **Clock Skew and Setup Checking Terminology**

Terms	Definition
constraint	The required time interval for the path, either specified explicitly by you with a FROM TO constraint, or derived from a PERIOD constraint.
Tpath	The summation of component and connection delays along the path.
Tsu (setup)	The setup requirement for the destination register.
Tsk (skew)	The difference between the arrival time for the destination register and the source register.
Slack	The negative slack shows that a setup error may occur, because the data from the source register does not set up at the target register for a subsequent clock edge.

register D (TclkD) less the delay from the clock input (CLKIOB) to register S (TclkS). Negative skew relative to the destination reduces the amount of time available for the data path, while positive skew relative to the destination register increases the amount of time available for the data path.

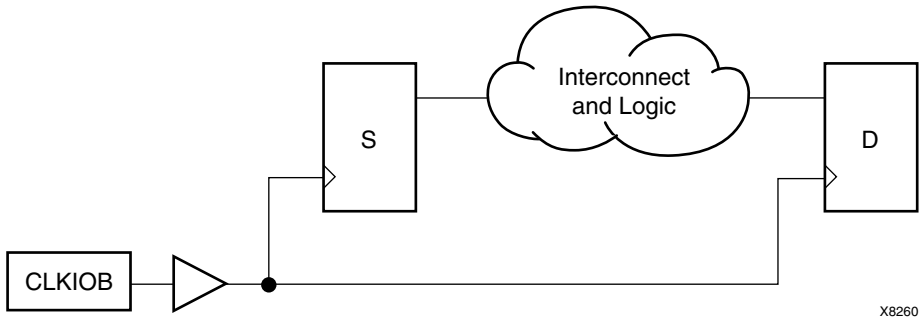


Figure 12-2: **Clock Skew Example**

Because the total clock path delay determines the clock arrival times at the source register (TclkS) and the destination register (TclkD), this check still applies if the source and destination clocks originate at the same chip input but travel through different clock buffers and routing resources, as shown in the following figure.

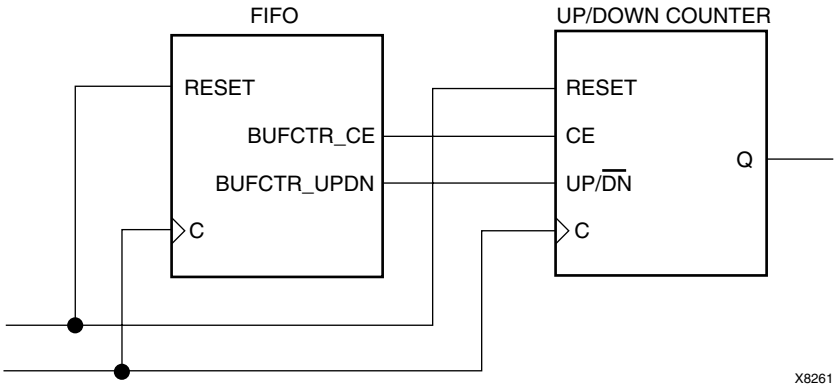


Figure 12-3: **Clock Passing Through Multiple Buffers**

When the source and destination clocks originate at different chip inputs, no obvious relationship between the two clock inputs exists for TRACE (because the software cannot determine the clock arrival time or phase information).

For FROM TO constraints, TRACE assumes you have taken into account the external timing relationship between the chip inputs. TRACE assumes both clock inputs arrive simultaneously. The difference between the destination clock arrival time (TclkD) and the source clock arrival time (TclkS) does not account for any difference in the arrival times at the two different clock inputs to the chip, as shown in the following figure.

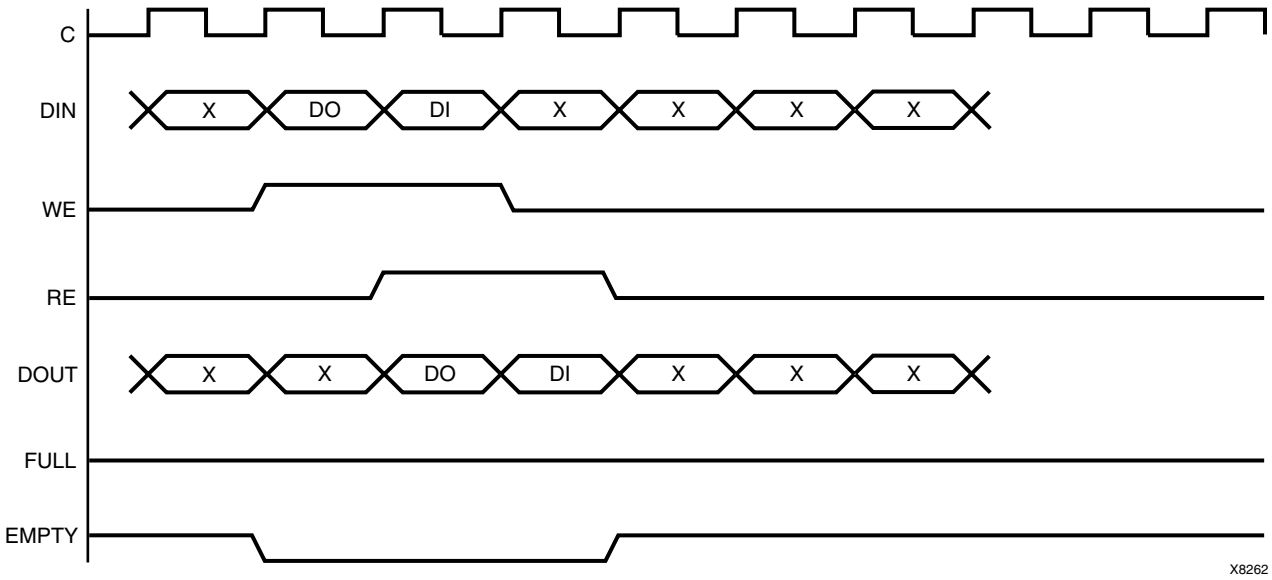


Figure 12-4: **Clocks Originating at Different Device Inputs**

The clock skew Tsk is not accounted for in setup checks covered by PERIOD constraints where the clock paths to the source and destination registers originate at different clock inputs.

Reporting with TRACE

The timing report produced by TRACE is a formatted ASCII (TWR) file prepared for a particular design. It reports statistics on the design, a summary of timing warnings and errors, and optional detailed net and path delay reports. The ASCII (TWR) reports are formatted for viewing in a monospace (non-proportional) font. If the text editor you use for viewing the reports uses a proportional font, the columns in the reports do not line up correctly.

In addition to the TWR file, you can generate an XML timing report (TWX) file using the –xml option. The contents of the XML timing report are identical to the ASCII (TWR) timing report, although the XML report is not formatted and can only be viewed with the Timing Analyzer GUI tool.

This section describes the following types of timing reports generated by TRACE.

- Summary Report—Lists summary information, design statistics, and statistics for each constraint in the PCF.
- Error Report—Lists timing errors and associated net/path delay information.
- Verbose Report—Lists delay information for all nets and paths.

In each type of report, the header specifies the command line used to generate the report, the type of report, the input design name, the optional input physical constraints file name, speed file version, and device and speed data for the input NCD file. At the end of each report is a timing summary, which includes the following information:

- The number of timing errors found in the design. This information appears in all reports.
- A timing score, showing the total amount of error (in picoseconds) for all timing constraints in the design.
- The number of paths and nets covered by the constraints.
- The number of route delays and the percentage of connections covered by timing constraints.

Note: The percentage of connections covered by timing constraints is given in a "% coverage" statistic. The statistic does not show the percentage of paths covered; it shows the percentage of connections covered. Even if you have entered constraints that cover all paths in the design, this percentage may be less than 100%, because some connections are never included for static timing analysis (for example, connections to the STARTUP component).

In the following sections, a description of each report is accompanied by a sample.

The following is a list of additional information on timing reports:

- For all timing reports, if you specify a physical constraints file that contains invalid data, a list of physical constraints file errors appears at the beginning of the report. These include errors in constraint syntax.
- In a timing report, a tilde (~) preceding a delay value shows that the delay value is approximate. Values with the tilde cannot be calculated exactly because of excessive delays, resistance, or capacitance on the net, that is, the path is too complex to calculate accurately.

The tilde (~) also means that the path may exceed the numerical value listed next to the tilde by as much as 20%. You can use the PENALIZE TILDE constraint to penalize these delays by a specified percentage (see the *Constraints Guide* for a description of the PENALIZE TILDE constraint).

- In a timing report, an "e" preceding a delay value shows that the delay value is estimated because the path is not routed.
- TRACE detects when a path cycles (that is, when the path passes through a driving output more than once), and reports the total number of cycles detected in the design. When TRACE detects a cycle, it disables the cycle from being analyzed. If the cycle itself is made up of many possible routes, each route is disabled for all paths that converge through the cycle in question and the total number is included in the reported cycle tally.

  A path is considered to cycle outside of the influence of other paths in the design. Thus, if a valid path follows a cycle from another path, but actually converges at an input and not a driving output, the path is not disabled and contains the elements of the cycle, which may be disabled on another path.
- Error counts reflect the number of path endpoints (register setup inputs, output pads) that fail to meet timing constraints, not the number of paths that fail the specification, as shown in the following figure.

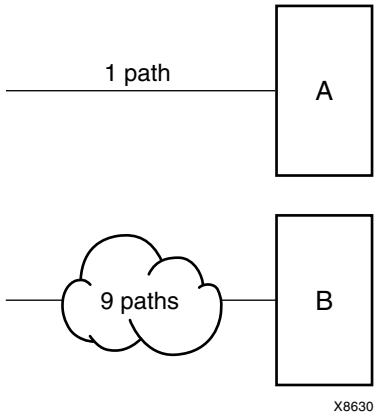


Figure 12-5: **Error reporting for failed timing constraints**

If an error is generated at the endpoints of A and B, the timing report would lists two errors—one for each endpoint.

- In Virtex-II designs, the MAP program places information that identifies dedicated clocks in the PCF. You must use a physical constraints file (PCF) generated by the MAP program to ensure accurate timing analysis on these designs.

Data Sheet Report

The Data Sheet report summarizes the external timing parameters for your design. Only inputs, outputs and clocks that have constraints appear in the Data Sheet report for verbose and error reports. Tables shown in the Data Sheet report depend on the type of timing paths present in the design, as well as the applied timing constraints. Unconstrained path analysis can be used with a constraints file to increase the coverage of the report to include paths not explicitly specified in the constraints file. In he absence of a physical constraints file (PCF), all I/O timing is analyzed and reported (less the effects of any default path tracing controls). The Data Sheet report includes the source and destination PAD names, and either the propagation delay between the source and destination or the setup and hold requirements for the source relative to the destination.

There are four methods of running TRACE to obtain a complete Data Sheet report:

- Run with advanced analysis (–a)
- Run using default analysis (that is, with no constraints file and without advanced analysis)
- Construct constraints to cover all paths in the design
- Run using the unconstrained path report for constraints that only partially cover the design

Following are tables, including delay characteristics, that appear in the Data Sheet report:

- Input Setup and Hold Times

  This table shows the setup and hold time for input signals with respect to an input clock at a source pad. It does not take into account any phase introduced by the DCM/DLL. If an input signal goes to two different destinations, the setup and hold are worst case for that signal. It might be the setup time for one destination and the hold time for another destination.

- Output Clock to Out Times

  This table shows the clock-to-out signals with respect to an input clock at a source pad. It does not take into account any phase introduced by the DCM/DLL. If an output signal is a combinatorial result of different sources that are clocked by the same clock, the clock-to-out is the worst-case path.

- Clock Table

  The clock table shows the relationship between different clocks. The Source Clock column shows all of the input clocks. The second column shows the delay between the rising edge of the source clock and the destination clock. The next column is the data delay between the falling edge of the source and the rising edge of the destination.

  If there is one destination flip-flop for each source flip-flop the design is successful. If a source goes to different flip-flops of unrelated clocks, one flip-flop might get the data and another flip-flop might miss it because of different data delays.

  You can quickly navigate to the Data Sheet report by clicking the corresponding item in the Hierarchical Report Browser.

- External Setup and Hold Requirements

  Timing accounts for clock phase relationships and DCM phase shifting for all derivatives of a primary clock input, and report separate data sheet setup and hold requirements for each primary input. Relative to all derivatives of a primary clock input covered by a timing constraint.

The maximum setup and hold times of device data inputs are listed relative to each clock input. When two or more paths from a data input exist relative to a device clock input, the worst-case setup and hold times are reported. One worst-case setup and hold time is reported for each data input and clock input combination in the design.

Following is an example of an external setup/hold requirement in the data sheet report:

```
Setup/Hold to clock ck1_i
------------+-------- -+----------+
            | Setup to  |Hold to   |
Source Pad  |clk (edge) |clk (edge)|
------------+-----------+----------+
start_i      |2.816(R)   |0.000(R)  |
-------------+-----------+----------+
```

- User-Defined Phase Relationships

  Timing reports separate setup and hold requirements for user-defined internal clocks in the data sheet report. User-defined external clock relationships are not reported separately.

- Clock-to-Clock Setup and Hold Requirements

  Timing will not report separate setup and hold requirements for internal clocks.

- Guaranteed Setup and Hold

  Guaranteed setup and hold requirements in the speed files will supersede any calculated setup and hold requirements made from detailed timing analysis. Timing will not include phase shifting, DCM duty cycle distortion, and jitter into guaranteed setup and hold requirements.

- Synchronous Propagation Delays

  Timing accounts for clock phase relationships and DCM phase shifting for all primary outputs with a primary clock input source, and reports separate clock-to-output and maximum propagation delay ranges for each primary output covered by a timing constraint.

  The maximum propagation delay from clock inputs to device data outputs are listed for each clock input. When two or more paths from a clock input to a data output exist, the worst-case propagation delay is reported. One worst-case propagation delay is reported for each data output and clock input combination.

Following is an example of clock-to-output propagation delays in the data sheet report:

```
Clock ck1_i to Pad
------------ ---+----------+
               |clk (edge)|
Destination Pad |to PAD    |
------------- --+----------+
out1_o          | 16.691(R)|
------------- --+----------+
Clock to Setup on destination clock ck2_i
----- -----+-------+--------+--------+--------+
           |Src/Dest |Src/Dest | Src/Dest| Src/Dest|
Source Clock|Rise/Rise|Fall/Rise|Rise/Fall|Fall/Fall|
-----------+-------+--------+--------+--------+
ck2_i      | 12.647 |        |       |        |
ck1_i      |10.241 |        |         |        |
-- --------+-------+--------+--------+--------+
```

The maximum propagation delay from each device input to each device output is reported if a combinational path exists between the device input and output. When two or more paths exist between a device input and output, the worst-case propagation delay is reported. One worst-case propagation delay is reported for every input and output combination in the design.

Following are examples of input-to-output propagation delays:

```
Pad to Pad
-----------------------------------------------
Source Pad   |Destination Pad|Delay  |
-------------+---------------+-------+
BSLOT0       |D0S            |37.534 |
BSLOT1       |D09            |37.876 |
BSLOT2       |D10            |34.627 |
BSLOT3       |D11            |37.214 |
CRESETN      |VCASN0         |51.846 |
CRESETN      |VCASN1         |51.846 |
CRESETN      |VCASN2         |49.776 |
CRESETN      |VCASN3         |52.408 |
CRESETN      |VCASN4         |52.314 |
CRESETN      |VCASN5         |52.314 |
CRESETN      |VCASN6           |51.357 |
CRESETN      |VCASN7           |52.527 |
-------------+-------------+---------
```

- User-Defined Phase Relationships

  Timing separates clock-to-output and maximum propagation delay ranges for user-defined internal clocks in the data sheet report. User-defined external clock relationships shall not be reported separately. They are broken out as separate external clocks.

Report Legend

The following table lists descriptions of what **X**, **R**, and **F** mean in the data sheet report.

Note: Applies to FPGA designs only.

X	Indeterminate
R	Rising Edge
F	Falling Edge

Guaranteed Setup and Hold Reporting

Guaranteed setup and hold values obtained from speed files are used in the data sheet reports for IOB input registers when these registers are clocked by specific clock routing resources and when the guaranteed setup and hold times are available for a specified device and speed.

Specific clock routing resources are clock networks that originate at a clock IOB, use a clock buffer to reach a clock routing resource and route directly to IOB registers.

Guaranteed setup and hold times are also used for reporting of input OFFSET constraints.

The following figure and text describes the external setup and hold time relationships.

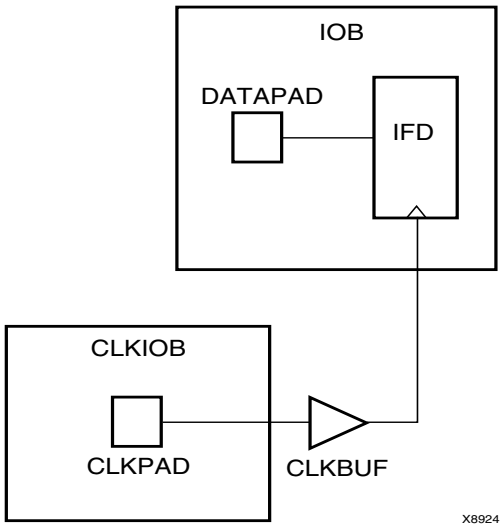


Figure 12-6: **Guaranteed Setup and Hold**

The pad CLKPAD of clock input component CLKIOB drives a global clock buffer CLKBUF, which in turn drives an input flip-flop IFD. The input flip-flop IFD clocks a data input driven from DATAPAD within the component IOB.

Setup Times

The external setup time is defined as the setup time of DATAPAD within IOB relative to CLKPAD within CLKIOB. When a guaranteed external setup time exists in the speed files for a particular DATAPAD and the CLKPAD pair and configuration, this number is used in timing reports. When no guaranteed external setup time exists in the speed files for a particular DATAPAD and CLKPAD pair, the external setup time is reported as the maximum path delay from DATAPAD to the IFD plus the maximum IFD setup time, less the minimum of maximum path delay(s) from the CLKPAD to the IFD.

Hold Times

The external hold time is defined as the hold time of DATAPAD within IOB relative to CLKPAD within CLKIOB. When a guaranteed external hold time exists in the speed files for a particular DATAPAD and the CLKPAD pair and configuration, this number is used in timing reports.

When no guaranteed external hold time exists in the speed files for a particular DATAPAD and CLKPAD pair, the external hold time is reported as the maximum path delay from CLKPAD to the IFD plus the maximum IFD hold time, less the minimum of maximum path delay(s) from the DATAPAD to the IFD.

Summary Report

The summary report includes the name of the design file being analyzed, the device speed and report level, followed by a statistical brief that includes the summary information and design statistics. The report also list statistics for each constraint in the PCF, including the number of timing errors for each constraint.

A summary report is produced when you do not enter an –e (error report) or –v (verbose report) option on the TRACE command line.

Two sample summary reports are shown below. The first sample shows the results without having a physical constraints file. The second sample shows the results when a physical constraints file is specified.

If no physical constraints file exists or if there are no timing constraints in the PCF, TRACE performs default path and net enumeration to provide timing analysis statistics. Default path enumeration includes all circuit paths to data and clock pins on sequential components and all data pins on primary outputs. Default net enumeration includes all nets.

Summary Report (Without a Physical Constraints File Specified)

The following sample summary report represents the output of this TRACE command.

trce –o summary.twr ramb16_s1.ncd

The name of the report is summary.twr. No preference file is specified on the command line, and the directory containing the file ram16_s1.ncd did not contain a PCF called ramb16_s1.pcf.

```
--------------------------------------------------------------------
Xilinx TRACE
Copyright (c) 1995-2005 Xilinx, Inc.  All rights reserved.
Design file:              ramb16_s1.ncd
Device,speed:             xc2v250,-6
Report level:             summary report
--------------------------------------------------------------------

WARNING:Timing - No timing constraints found, doing default
enumeration.
Asterisk (*) preceding a constraint indicates it was not met.
--------------------------------------------------------------------
  Constraint                      | Requested  | Actual | Logic

                                  |      |                | Levels
--------------------------------------------------------------------
  Default period analysis         |            | 2.840ns   | 2
--------------------------------------------------------------------
  Default net enumeration           |            | 0.001ns |
--------------------------------------------------------------------

All constraints were met.

Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)

Setup/Hold to clock clk
---------------+------------+------------+
               |  Setup to  |  Hold to   |
Source Pad     | clk (edge) | clk (edge) |
---------------+------------+------------+
ad0            |    0.263(R)|    0.555(R)|
ad1            |    0.263(R)|    0.555(R)|
ad10           |    0.263(R)|    0.555(R)|
ad11           |    0.263(R)|    0.555(R)|
ad12           |    0.263(R)|    0.555(R)|
ad13           |    0.263(R)|    0.555(R)|
.
.
.
---------------+------------+------------+
Clock clk to Pad
---------------+------------+
               | clk (edge) |
Destination Pad|   to PAD   |
---------------+------------+
d0             |    7.496(R)|
---------------+------------+
```

```
Timing summary:
---------------
Timing errors: 0  Score: 0

Constraints cover 20 paths, 21 nets, and 21 connections (100.0%
coverage)

Design statistics:
   Minimum period:   2.840ns (Maximum frequency: 352.113MHz)
   Maximum combinational path delay:   6.063ns
   Maximum net delay:   0.001ns
Analysis completed Wed Mar  8 14:52:30 2000
--------------------------------------------------------------------
Summary Report (With a Physical Constraints File Specified)
The following sample summary report represents the output of this TRACE
command:
trce -o summary1.twr ramb16_s1.ncd clkperiod.pcf
The name of the report is summary1.twr. The timing analysis represented
in the file were performed by referring to the constraints in the file
clkperiod.pcf.
--------------------------------------------------------------------
Xilinx TRACE
Copyright (c) 1995-2005 Xilinx, Inc.  All rights reserved.

Design file:              ramb16_s1.ncd
Physical constraint file: clkperiod.pcf
Device,speed:             xc2v250,-6
Report level:             summary report
--------------------------------------------------------------------

Asterisk (*) preceding a constraint indicates it was not met.

--------------------------------------------------------------------
Constraint                          | Requested | Actual    | Logic
                                    |           |           | Levels
--------------------------------------------------------------------
TS01 = PERIOD TIMEGRP "clk" 10.0ns|         |         |
--------------------------------------------------------------------
OFFSET = IN 3.0 ns AFTER COMP
"clk" TIMEG                         | 3.000ns   | 8.593ns |2
RP "rams"
--------------------------------------------------------------------
* TS02 = MAXDELAY FROM TIMEGRP
"rams" TO TI                        | 6.000ns   | 6.063ns  |2
  MEGRP "pads" 6.0 ns|              |           |
--------------------------------------------------------------------
1 constraint not met.

Data Sheet report:
------------------
All values displayed in nanoseconds (ns)
```

```
Setup/Hold to clock clk
---------------+------------+------------+
               |  Setup to  |  Hold to   |
Source Pad     | clk (edge) | clk (edge) |
---------------+------------+------------+
ad0            |    0.263(R)|    0.555(R)|
ad1            |    0.263(R)|    0.555(R)|
ad10           |    0.263(R)|    0.555(R)|
ad11           |    0.263(R)|    0.555(R)|
ad12           |    0.263(R)|    0.555(R)|
ad13           |    0.263(R)|    0.555(R)|
.
.
.
---------------+------------+------------+
Clock clk to Pad
---------------+------------+
               | clk (edge) |
Destination Pad|   to PAD   |
---------------+------------+
d0             |    7.496(R)|
---------------+------------+

Timing summary:
---------------

Timing errors: 1  Score: 63

Constraints cover 19 paths, 0 nets, and 21 connections (100.0% coverage)

Design statistics:
   Maximum path delay from/to any node:   6.063ns
   Maximum input arrival time after clock:   8.593ns

Analysis completed Wed Mar  8 14:54:31 2005
------------------------------------------------------------------------
```

When the physical constraints file includes timing constraints, the summary report lists the percentage of all design connections covered by timing constraints. If there are no timing constraints, the report shows 100% coverage. An asterisk (*) precedes constraints that fail.

Error Report

The error report lists timing errors and associated net and path delay information. Errors are ordered by constraint in the PCF and within constraints, by slack (the difference between the constraint and the analyzed value, with a negative slack showing an error condition). The maximum number of errors listed for each constraint is set by the limit you enter on the command line. The error report also contains a list of all time groups defined in the PCF and all of the members defined within each group.

The main body of the error report lists all timing constraints as they appear in the input PCF. If the constraint is met, the report states the number of items scored by TRACE, reports no timing errors detected, and issues a brief report line, showing important information (for example, the maximum delay for the particular constraint). If the constraint is not met, it gives the number of items scored by TRACE, the number of errors encountered, and a detailed breakdown of the error.

For errors in which the path delays are broken down into individual net and component delays, the report lists each physical resource and the logical resource from which the physical resource was generated.

As in the other three types of reports, descriptive material appears at the top. A timing summary always appears at the end of the reports.

The following sample error report (error.twr) represents the output generated with this TRACE command:

```
trce -e 3 ramb16_s1.ncd clkperiod.pcf -o error_report.twr

--------------------------------------------------------------------
Xilinx TRACE
Copyright (c) 1995-2005 Xilinx, Inc.  All rights reserved.

trce -e 3 ramb16_s1.ncd clkperiod.pcf -o error_report.twr

Design file:              ramb16_s1.ncd
Physical constraint file: clkperiod.pcf
Device,speed:             xc2v250,-5 (ADVANCED 1.84 2001-05-09)
Report level:             error report
--------------------------------------------------------------------


====================================================================
Timing constraint: TS01 = PERIOD TIMEGRP "clk" 10.333ns ;

 0 items analyzed, 0 timing errors detected.
--------------------------------------------------------------------

====================================================================
Timing constraint: OFFSET = IN 3.0 ns AFTER COMP "clk" TIMEGRP "rams" ;

 18 items analyzed, 0 timing errors detected.
 Maximum allowable offset is   9.224ns.
--------------------------------------------------------------------

====================================================================
Timing constraint: TS02 = MAXDELAY FROM TIMEGRP "rams" TO TIMEGRP "pads"
8.0 nS  ;

 1 item analyzed, 1 timing error detected.
 Maximum delay is   8.587ns.
--------------------------------------------------------------------
Slack:                  -0.587ns (requirement - data path)
  Source:               RAMB16.A
  Destination:          d0
  Requirement:          8.000ns
  Data Path Delay:      8.587ns (Levels of Logic = 2)
  Source Clock:         CLK rising at 0.000ns
```

```
  Data Path: RAMB16.A to d0
  Location    Delay type  Delay(ns) Physical Resource
                                    Logical Resource(s)
  ----------------------------------------------------
  RAMB16.DOA0 Tbcko         3.006   RAMB16
                                    RAMB16.A
  IOB.O1      net         e 0.100   N$41
              (fanout=1)
  IOB.PAD     Tioop         5.481   d0
                                    I$22
                                    d0
  ----------------------------------------------------
  Total                     8.587ns (8.487ns logic, 0.100ns
                                    route)
                                   (98.8% logic, 1.2%
                                    route)

--------------------------------------------------------------------------------

1 constraint not met.

Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)

Setup/Hold to clock clk
---------------+------------+------------+
               |  Setup to  |  Hold to   |
Source Pad     | clk (edge) | clk (edge) |
---------------+------------+------------+
ad0            |   -0.013(R)|    0.325(R)|
ad1            |   -0.013(R)|    0.325(R)|
ad10           |   -0.013(R)|    0.325(R)|
ad11           |   -0.013(R)|    0.325(R)|
ad12           |   -0.013(R)|    0.325(R)|
ad13           |   -0.013(R)|    0.325(R)|
.
.
.
---------------+------------+------------+
```

```
Clock clk to Pad
---------------+------------+
               | clk (edge) |
Destination Pad|   to PAD   |
---------------+------------+
d0             |    9.563(R)|
---------------+------------+


Timing summary:
---------------

Timing errors: 1  Score: 587

Constraints cover 19 paths, 0 nets, and 21 connections (100.0% coverage)

Design statistics:
   Maximum path delay from/to any node:   8.587ns
   Maximum input arrival time after clock:   9.224ns

Analysis completed Mon Jun 03 17:47:21 2005
--------------------------------------------------------------------
```

Verbose Report

The verbose report is similar to the error report and provides details on delays for all constrained paths and nets in the design. Entries are ordered by constraint in the PCF, which may differ from the UCF or NCF and, within constraints, by slack, with a negative slack showing an error condition. The maximum number of items listed for each constraint is set by the limit you enter on the command line.

Note: The data sheet report and STAMP model display skew values on non-dedicated clock resources that do not display in the default period analysis of the normal verbose report. The data sheet report and STAMP model must include skew because skew affects the external timing model. To display skew values in the verbose report, use the –skew option.

The verbose report also contains a list of all time groups defined in the PCF, and all of the members defined within each group.

The body of the verbose report enumerates each constraint as it appears in the input physical constraints file, the number of items scored by TRACE for that constraint, and the number of errors detected for the constraint. Each item is described, ordered by descending slack. A Report line for each item provides important information, such as the amount of delay on a net, fanout on each net, location if the logic has been placed, and by how much the constraint is met.

For path constraints, if there is an error, the report shows the amount by which the constraint is exceeded. For errors in which the path delays are broken down into individual net and component delays, the report lists each physical resource and the logical resource from which the physical resource was generated.

If there are no errors, the report shows that the constraint passed and by how much. Each logic and route delay is analyzed, totaled, and reported.

The following sample verbose report (verbose.twr) represents the output generated with this TRACE command:

```
trce –v 1 ramb16_s1.ncd clkperiod.pcf –o verbose_report.twr
```

```
--------------------------------------------------------------------
Xilinx TRACE
Copyright (c) 1995-2005 Xilinx, Inc.  All rights reserved.

trce -v 1 ramb16_s1.ncd clkperiod.pcf -o verbose_report.twr

Design file:              ramb16_s1.ncd
Physical constraint file: clkperiod.pcf
Device,speed:             xc2v250,-5 (ADVANCED 1.84 2001-05-09)
Report level:          verbose report, limited to 1 item per constraint
--------------------------------------------------------------------


====================================================================
Timing constraint: TS01 = PERIOD TIMEGRP "clk" 10.333ns ;

 0 items analyzed, 0 timing errors detected.
--------------------------------------------------------------------

====================================================================
Timing constraint: OFFSET = IN 3.0 ns AFTER COMP "clk" TIMEGRP "rams" ;

 18 items analyzed, 0 timing errors detected.
 Maximum allowable offset is   9.224ns.
--------------------------------------------------------------------
Slack:                  6.224ns (requirement - (data path - clock path
- clock arrival))
  Source:               ssr
  Destination:          RAMB16.A
  Destination Clock:    CLK rising at 0.000ns
  Requirement:          7.333ns
  Data Path Delay:      2.085ns (Levels of Logic = 2)
  Clock Path Delay:     0.976ns (Levels of Logic = 2)

  Data Path: ssr to RAMB16.A
  Location    Delay type  Delay(ns)
  Physical Resource
                                    Logical Resource(s)
  ------------------------------------------------------
  IOB.I       Tiopi
       0.551     ssr
                           ssr
                           I$36
  RAM16.SSRA  net       e 0.100     N$9
              (fanout=1)
  RAM16.CLKA  Tbrck       1.434      RAMB16
                                    RAMB16.A
  ------------------------------------------------------
  Total                     2.085ns (1.985ns logic, 0.100ns
                                    route)
                                    (95.2% logic, 4.8%
                                    route)
```

```
  Clock Path: clk to RAMB16.A
  Location    Delay type  Delay(ns)  Physical Resource
                                     Logical Resource(s)
  -------------------------------------------------------
  IOB.I       Tiopi       0.551      clk
                                     clk
                                     clk/new_buffer
  BUFGMUX.I0  net       e 0.100      clk/new_buffer
              (fanout=1)
  BUFGMUX.O   Tgi0o       0.225      I$9
                                     I$9
  RAM16.CLKA  net       e 0.100      CLK
              (fanout=1)

  -------------------------------------------------------
  Total                     0.976ns  (0.776ns logic, 0.200ns
                                     route)
                                     (79.5% logic, 20.5%
                                     route)

--------------------------------------------------------------------------------

================================================================================
Timing constraint: TS02 = MAXDELAY FROM TIMEGRP "rams" TO TIMEGRP "pads"
8.0 nS  ;

 1 item analyzed, 1 timing error detected.
 Maximum delay is   8.587ns.
--------------------------------------------------------------------------------
Slack:                  -0.587ns (requirement - data path)
  Source:               RAMB16.A
  Destination:          d0
  Requirement:          8.000ns
  Data Path Delay:      8.587ns (Levels of Logic = 2)
  Source Clock:         CLK rising at 0.000ns

  Data Path: RAMB16.A to d0
    Location            Delay type         Delay(ns)  Physical Resource
                                                      Logical Resource(s)
    ----------------------------------------------------  -----------
    RAMB16.DOA0          Tbcko                 3.006   RAMB16
                                                       RAMB16.A
    IOB.O1               net (fanout=1)     e  0.100   N$41
    IOB.PAD              Tioop                 5.481   d0
                                                       I$22
                                                       d0
    ----------------------------------------------------  -----------
    Total                                      8.587ns (8.487ns logic,
0.100ns route)
                                              (98.8% logic, 1.2% route)

--------------------------------------------------------------------------------

1 constraint not met.
```

```
Data Sheet report:
------------------
All values displayed in nanoseconds (ns)

Setup/Hold to clock clk
---------------+------------+------------+
               |  Setup to  |  Hold to   |
Source Pad     | clk (edge) | clk (edge) |
---------------+------------+------------+
ad0            |   -0.013(R)|    0.325(R)|
ad1            |   -0.013(R)|    0.325(R)|
ad10           |   -0.013(R)|    0.325(R)|
ad11           |   -0.013(R)|    0.325(R)|
.
.
.
---------------+------------+------------+

Clock clk to Pad
---------------+------------+
               | clk (edge) |
Destination Pad|   to PAD   |
---------------+------------+
d0             |    9.563(R)|
---------------+------------+


Timing summary:
---------------

Timing errors: 1  Score: 587

Constraints cover 19 paths, 0 nets, and 21 connections (100.0% coverage)

Design statistics:
   Maximum path delay from/to any node:   8.587ns
   Maximum input arrival time after clock:   9.224ns


Analysis completed Mon Jun 03 17:57:24 2005
--------------------------------------------------------------------
```

OFFSET Constraints

OFFSET constraints define Input and Output timing constraints with respect to an initial time of 0ns.

The associated PERIOD constraint defines the initial clock edge. If the PERIOD constraint is defined with the attribute HIGH, the initial clock edge is the rising clock edge. If the attribute is LOW, the initial clock edge is the falling clock edge. This can be changed by using the HIGH/LOW keyword in the OFFSET constraint. The OFFSET constraint checks the setup time and hold time. For additional information on timing constraints, please refer to the Constraints Guide at http://www.xilinx.com/support under Documentation, Software Manuals.

OFFSET IN Constraint Examples

This section describes in detail a specific example of an OFFSET IN constraint as shown in the Timing Constraints section of a timing analysis report. For clarification, the OFFSET IN constraint information is divided into the following parts:

- OFFSET IN Header
- OFFSET IN Path Details
- OFFSET IN Detailed Path Data
- OFFSET IN Detail Path Clock Path
- OFFSET IN with Phase Clock

OFFSET IN Header

The header includes the constraint, the number of items analyzed, and number of timing errors detected. Please see PERIOD Header for more information on items analyzed and timing errors.

```
Example:
================================================================================

Timing constraint: OFFSET = IN 4 nS  BEFORE COMP "wclk_in" ;

113 items analyzed, 30 timing errors detected.

Minimum allowable offset is   4.468ns.


--------------------------------------------------------------------------------
```

The minimum allowable offset is 4.468 ns. Because this is an OFFSET IN BEFORE, it means the data must be valid 4.468 ns before the initial edge of the clock. The PERIOD constraint was defined with the keyword HIGH, therefore the initial edge of the clock is the rising edge.

OFFSET IN Path Details

This path fails the constraint by 0.468 ns. The slack equation shows how the slack was calculated. In respect to the slack equation data delay increases the setup time while clock delay decreases the setup time. The clock arrival time is also taken into account. In this example, the clock arrival time is 0.000 ns; therefore, it does not affect the slack.

```
Example:

================================================================================

Slack:                  -0.468ns (requirement - (data path - clock path
- clock arrival + uncertainty))

  Source:                wr_enl (PAD)

 Destination:           wr_addr[2] (FF)

  Destination Clock:     wclk rising at 0.000ns

  Requirement:           4.000ns
```

```
  Data Path Delay:      3.983ns (Levels of Logic = 2)
  Clock Path Delay:     -0.485ns (Levels of Logic = 3)
  Clock Uncertainty:    0.000ns
  Data Path: wr_enl to wr_addr[2]
--------------------------------------------------------------------------
```

OFFSET IN Detailed Path Data

The first section is the data path. In the following case, the path starts at an IOB, goes through a look-up table (LUT) and is the clock enable pin of the destination flip-flop.

Example:

```
--------------------------------------------------------------------------
 Data Path: wr_enl to wr_addr[2]
   Location            Delay type         Delay(ns)  Logical Resource(s)
   -------------------------------------------------  -----------------
    C4.I                 Tiopi                 0.825   wr_enl
                                                       wr_enl_ibuf
    SLICE_X2Y9.G3        net (fanout=39)       1.887   wr_enl_c
    SLICE_X2Y9.Y         Tilo                  0.439   G_82
    SLICE_X3Y11.CE       net (fanout=1)        0.592   G_82
    SLICE_X3Y11.CLK      Tceck                 0.240   wr_addr[2]

    ----------------------------------------------------  ---------------
    Total                    3.983ns (1.504ns logic, 2.479ns route)
                                        37.8% logic, 62.2% route)

--------------------------------------------------------------------------
```

OFFSET IN Detail Path Clock Path

The second section is the clock path. In this example the clock starts at an IOB, goes to a DCM, comes out CLK0 of the DCM through a global buffer (BUFGHUX). It ends at a clock pin of a FF.

The Tdcmino is a calculated delay. This is the equation:

```
Clock Path: wclk_in to wr_addr[2]

    Location            Delay type         Delay(ns)  Logical Resource(s)

    -------------------------------------------------  -----------------

     D7.I                 Tiopi                 0.825   wclk_in

                                                        write_dcm/IBUFG

     DCM_X0Y1.CLKIN       net (fanout=1)        0.798   write_dcm/IBUFG

     DCM_X0Y1.CLK0        Tdcmino              -4.297   write_dcm/CLKDLL

     BUFGMUX3P.I0         net (fanout=1)        0.852   write_dcm/CLK0

     BUFGMUX3P.O          Tgi0o                 0.589   write_dcm/BUFG

     SLICE_X3Y11.CLK      net (fanout=41)       0.748   wclk

    -------------------------------------------------  -----------------

    Total                                     -0.485ns (-2.883ns logic,
2.398ns route)

--------------------------------------------------------------------------
```

OFFSET In with Phase Shifted Clock

In the following example, the clock is the CLK90 output of the DCM. The clock arrival time is 2.5 ns. The rclk_90 rising at 2.500 ns. This number is calculated from the PERIOD on rclk_in which is 10ns in this example. The 2.5 ns affects the slack. Because the clock is delayed by 2.5 ns, the data has 2.5 ns longer to get to the destination.

If this path used the falling edge of the clock, the destination clock would say, falling at 00 ns 7.500 ns (2.5 for the phase and 5.0 for the clock edge). The minimum allowable offset can be negative because it is relative to the initial edge of the clock. A negative minimum allowable offset means the data can arrive after the initial edge of the clock. This often occurs when the destination clock is falling while the initial edge is defined as rising. This can also occur on clocks with phase shifting.

```
Example:

==========================================================================

Timing constraint: OFFSET = IN 4 nS  BEFORE COMP "rclk_in" ;

2 items analyzed, 0 timing errors detected.

Minimum allowable offset is   1.316ns.
```

```
--------------------------------------------------------------------------------
Slack:                  2.684ns (requirement - (data path - clock path
- clock arrival + uncertainty))

Source:               wclk_in (PAD)

Destination:          ffl_reg (FF)

Destination Clock:    rclk_90 rising at 2.500ns

Requirement:          4.000ns

Data Path Delay:      3.183ns (Levels of Logic = 5)

Clock Path Delay:     -0.633ns (Levels of Logic = 3)

Clock Uncertainty:    0.000ns

Data Path: wclk_in to ffl_reg

Location             Delay type         Delay(ns)  Logical Resource(s)


-------------------------------------------------  -------------------

D7.I                 Tiopi                 0.825   wclk_in

                                                   write_dcm/IBUFG

DCM_X0Y1.CLKIN       net (fanout=1)        0.798   write_dcm/IBUFG

DCM_X0Y1.CLK0        Tdcmino              -4.297   write_dcm/CLKDLL

BUFGMUX3P.I0         net (fanout=1)        0.852   write_dcm/CLK0

BUFGMUX3P.O          Tgi0o                 0.589   write_dcm/BUFG

SLICE_X2Y11.G3       net (fanout=41)       1.884   wclk

SLICE_X2Y11.Y        Tilo                  0.439   un1_full_st

SLICE_X2Y11.F3       net (fanout=1)        0.035   un1_full_st

SLICE_X2Y11.X       Tilo                0.439   full_st_i_0.G_4.G_4.G_4

K4.O1                net (fanout=3)        1.230   G_4

K4.OTCLK1            Tioock                0.389   ffl_reg


-------------------------------------------------  -------------------
    Total                                     3.183ns (-1.616ns logic,
4.799ns route)
```

```
Clock Path: rclk_in to ffl_reg

Location             Delay type         Delay(ns)  Logical Resource(s)

-------------------------------------------------  -------------------

A8.I                 Tiopi                 0.825   rclk_in

                                                   read_ibufg

CM_X1Y1.CLKIN        net (fanout=1)        0.798   rclk_ibufg

CM_X1Y1.CLK90        Tdcmino              -4.290   read_dcm

UFGMUX5P.I0          net (fanout=1)        0.852   rclk_90_dcm

BUFGMUX5P.O           Tgi0o                 0.589   read90_bufg

4.OTCLK1             net (fanout=2)        0.593   rclk_90

-------------------------------------------------  -------------------

    Total -0.633ns (-2.876ns logic, 2.243ns route)

--------------------------------------------------------------------------
```

OFFSET OUT Constraint Examples

The following section describe specific examples of an OFFSET OUT constraint, as shown in the Timing Constraints section of a timing report. For clarification, the OFFSET OUT constraint information is divided into the following parts:

- OFFSET OUT Header
- OFFSET OUT Path Details
- OFFSET OUT Detail Clock Path
- OFFSET OUT Detail Path Data

OFFSET OUT Header

The header includes the constraint, the number of items analyzed, and number of timing errors detected. See the PERIOD Header for more information on items analyzed and timing errors.

```
Example:

==========================================================================

Timing constraint: OFFSET = OUT 10 nS  AFTER COMP "rclk_in" ;

 50 items analyzed, 0 timing errors detected.


 Minimum allowable offset is   9.835ns.

--------------------------------------------------------------------------
```

OFFSET OUT Path Details

The example path below passed the timing constraint by .533 ns. The slack equation shows how the slack was calculated. Data delay increases the clock to out time and clock delay also increases the clock to out time. The clock arrival time is also taken into account. In this example the clock arrival time is 0.000 ns; therefore, it does not affect the slack.

If the clock edge occurs at a different time, this value is also added to the clock to out time. If this example had the clock falling at 5.000 ns, 5.000 ns would be added to the slack equation because the initial edge of the corresponding PERIOD constraint is HIGH.

Note: The clock falling at 5.000 ns is determined by how the PERIOD constraint isdefined, for example PERIOD 10 HIGH 5.

```
Example:

==============================================================================
Slack:                  0.533ns (requirement - (clock arrival + clock
path + data path + uncertainty))

  Source:               wr_addr[2] (FF)

  Destination:          efl (PAD)

  Source Clock:         wclk rising at 0.000ns

  Requirement:          10.000ns

  Data Path Delay:      9.952ns (Levels of Logic = 4)

  Clock Path Delay:     -0.485ns (Levels of Logic = 3)

  Clock Uncertainty:    0.000ns

------------------------------------------------------------------------------
```

OFFSET OUT Detail Clock Path

In the following example, because the OFFSET OUT path starts with the clock, the clock path is shown first. The clock starts at an IOB, goes to a DCM, comes out CLK0 of the DCM through a global buffer. It ends at a clock pin of a FF.

The Tdcmino is a calculated delay. This is the equation:

```
Clock Path: rclk_in to rd_addr[2]

Location             Delay type         Delay(ns)  Logical
Resource(s)

-------------------------------------------------

    A8.I                Tiopi                 0.825   rclk_in

                                                      read_ibufg

    DCM_X1Y1.CLKIN      net (fanout=1)        0.798   rclk_ibufg

    DCM_X1Y1.CLK0       Tdcmino              -4.290   read_dcm

    BUFGMUX7P.I0        net (fanout=1)        0.852   rclk_dcm
```

```
    BUFGMUX7P.O          Tgi0o                 0.589   read_bufg

    SLICE_X4Y10.CLK      net (fanout=4)        0.738   rclk
    ------------------------------------------------

    Total                                     -0.488ns (-2.876ns
logic, 2.388ns route)
```

OFFSET OUT Detail Path Data

The second section is the data path. In this case, the path starts at an FF, goes through three look-up tables and ends at the IOB.

```
Example:

--------------------------------------------------------------------------

Data Path: rd_addr[2] to efl

   Location           Delay type        Delay(ns)  Logical Resource(s)

    -------------------------------------------------  ---------------

    SLICE_X4Y10.YQ       Tcko                  0.568   rd_addr[2]

    SLICE_X2Y10.F4       net (fanout=40)       0.681   rd_addr[2]

    SLICE_X2Y10.X        Tilo                  0.439   G_59

    SLICE_X2Y10.G1       net (fanout=1)        0.286   G_59

    SLICE_X2Y10.Y        Tilo                  0.439   N_44_i

    SLICE_X0Y0.F2        net (fanout=3)        1.348   N_44_i

    SLICE_X0Y0.X         Tilo                  0.439   empty_st_i_0

    M4.O1                net (fanout=2)        0.474   empty_st_i_0

    M4.PAD               Tioop                 5.649   efl_obuf

                                                       efl


    -------------------------------------------------  ---------------

    Total 10.323ns (7.534ns logic, 2.789ns route)

(73.0% logic, 27.0% route)
```

PERIOD Constraints

A PERIOD constraint identifies all paths between all sequential elements controlled by the given clock signal name. For additional information on timing constraints, please refer to the Constraints Guide.

PERIOD Constraints Examples

The following section provides examples and details of the PERIOD constraints shown in the Timing Constraints section of a timing analysis report. For clarification, PERIOD constraint information is divided into the following parts:

- PERIOD Header
- PERIOD Path
- PERIOD Path Details
- PERIOD Constraint with PHASE

PERIOD Header

This following example is of a constraint generated using NGDbuild during the translate step in the Xilinx design flow. A new timespec (constraint) name was created. In this example it is TS_write_dcm_CLK0. Write_dcm is the instantiated name of the DCM. CLK0 is the output clock. The timegroup created for the PERIOD constraint is write_dcm_CLK0. The constraint is related to TS_wclk. In this example, the PERIOD constraint is the same as the original constraint because the original constraint is multiplied by 1 and there is not a phase offset. Because TS_wclk is defined to have a Period of 12 ns, this constraint has a Period of 12 ns.

In this constraint, 296 items are analyzed. An item is a path or a net. Because this constraint deals with paths, an item refers to a unique path. If the design has unique paths to the same endpoints, this is counted as two paths. If this constraint were a MAXDELAY or a net-based constraint, items refer to nets. The number of timing errors refers to the number of endpoints that do not meet the timing requirement, and the number of endpoints with hold violations. If the number of hold violations is not shown, there are no hold violations for this constraint. If there are two or more paths to the same endpoint, it is considered one timing error. If this is the situation, the report shows two or more detailed paths; one for each path to the same endpoint.

The next line reports the minimum Period for this constraint, which is how fast this clock runs.

Example:

```
================================================================================

Timing constraint: TS_write_dcm_CLK0 = PERIOD TIMEGRP "write_dcm_CLK0"
TS_wclk *
1.000000 HIGH

50.000 % ;

296 items analyzed, 0 timing errors detected.

 Minimum period is   3.825ns.

--------------------------------------------------------------------------------
```

PERIOD Path

The detail path section shows all of the details for each path in the analyzed timing constraint. The most important thing it does is identify if the path meets the timing requirement. This information appears on the first line and is defined as the **Slack**. If the slack number is positive, the path meets timing constraint by the slack amount. If the slack number is negative, the path fails the timing constraint by the slack amount. Next to the slack number is the equation used for calculating the slack. The requirement is the time constraint number. In this case, it is 12 ns Because that is the time for the original timespec TS_wclk. The data path delay is 3.811 ns and the clock skew is negative 0.014 ns. (12 - (3.811 - 0.014) = 8.203). The detail paths are sorted by slack. The path with the least amount of slack, is the first path shown in the Timing Constraints section.

The **Source** is the starting point of the path. Following the source name is the type of component. In this case the component is a flip-flop (FF). The FF group also contains the SRL16. Other components are RAM (Distributed RAM vs BlockRAM), PAD, LATCH, HSIO (High Speed I/O such as the Gigabit Transceivers) MULT (Multipliers), CPU (PowerPC), and others. In Timing Analyzer, for FPGA designs the Source is a hot-link for cross probing. For more information on Cross Probing please see Cross Probing with Floorplanner.

The **Destination** is the ending point of the path. See the above description of the Source for more information about Destination component types and cross probing.

The **Requirement** is a calculated number based on the time constraint and the time of the clock edges. The source and destination clock of this path are the same so the entire requirement is used. If the source or destination clock was a related clock, the new requirement would be the time difference between the clock edges. If the source and destination clocks are the same clock but different edges, the new requirement would be half the original period constraint.

The **Data Path Delay** is the delay of the data path from the source to the destination. The levels of logic are the number of LUTS that carry logic between the source and destination. It does not include the clock-to-out or the setup at the destination. If there was a LUT in the same slice of the destination, that counts as a level of logic. For this path, there is no logic between the source and destination therefore the level of logic is 0.

The **Clock Skew** is the difference between the time a clock signal arrives at the source flip-flop in a path and the time it arrives at the destination flip-flop. If Clock Skew is not checked it will not be reported.

The **Source Clock** or the **Destination Clock** report the clock name at the source or destination point. It also includes if the clock edge is the rising or falling edge and the time that the edge occurs. If clock phase is introduced by the DCM/DLL, it would show up in the arrival time of the clock. This includes coarse phase (CLK90, CLK180, or CLK270) and fine phase introduced by Fixed Phase Shift or the initial phase of Variable Phase Shift

The **Clock Uncertainty** for an OFFSET constraint might be different than the clock uncertainty on a PERIOD constraint for the same clock. The OFFSET constraint only looks at one clock edge in the equation but the PERIOD constraints takes into account the uncertainty on the clock at the source registers and the uncertainty on the clock at the destination register therefore there are two clock edges in the equation.

```
Example:

--------------------------------------------------------------------------------

Slack:                  8.175ns (requirement - (data path - clock skew
+ uncertainty))

  Source:               wr_addr[0] (FF)

  Destination:          fifo_ram/BU5/SP (RAM)

  Requirement:          12.000ns

  Data Path Delay:      3.811ns (Levels of Logic = 1)

  clock skew:      -0.014ns

  Source Clock:         wclk rising at 0.000ns

  Destination Clock:    wclk rising at 12.000ns

  Clock Uncertainty:    0.000ns


--------------------------------------------------------------------------------
```

PERIOD Path Details

The first line is a link to the Constraint Improvement Wizard (CIW). The CIW gives suggestions for resolving timing constraint issues if it is a failing path. The data path section shows all the delays for each component and net in the path. The first column is the **Location** of the component in the FPGA. It is turned off by default in TWX reports. The next column is the **Delay Type**. If it is a net, the fanout is shown. The Delay names correspond with the datasheet. For an explanation of the delay names, click on a delay name for a description page to appear. Descriptions for Virtex-E , Virtex-II , Virtex-II Pro and Spartan-II architectures are available.

The next columns are the **Physical Resource** and **Logical Resource** names. The Physical name is the name of the component after mapping. This name is generated by the Map process. It is turned off by default in TWX reports. The logical name is the name in the design file. This is typically created by the synthesis tool or schematic capture program.

At the end of the path is the total amount of the delay followed by a break down of logic vs routing. This is useful information for debugging a timing failure. For more information see Timing Improvement Wizard for suggestions on how to fix a timing issues.

```
Example:

--------------------------------------------------------------------------------

Constraints Improvement Wizard
Data Path: wr_addr[0] to fifo_ram/BU5/SP

   Location           Delay type        Delay(ns)  Logical Resource(s)

    -------------------------------------------------  ---------------

    SLICE_X2Y4.YQ        Tcko                  0.568   wr_addr[0]


    SLICE_X6Y8.WF1       net (fanout=112)      2.721   wr_addr[0]

    SLICE_X6Y8.CLK       Tas                   0.522   fifo_ram/BU5/SP

    -------------------------------------------------  ---------------

    Total                                     3.811ns (1.090ns logic,
2.721ns route)

                                             (28.6% logic, 71.4% route)

--------------------------------------------------------------------------------
```

PERIOD Constraint with PHASE

This is a PERIOD constraint for a clock with Phase. It is a constraint created by the Translate (ngdbuild) step. It is related back to the TS_rclk constraint with a PHASE of 2.5 ns added. The clock is the CLK90 output of the DCM. Since the PERIOD constraint is 10 ns the clock phase from the CLK90 output is 2.5 ns, one-fourth of the original constraint. This is defined using the PHASE keyword.

```
Example:

Timing constraint: TS_rclk_90_dcm = PERIOD TIMEGRP "rclk_90_dcm"
TS_rclk * 1.000000 PHASE + 2.500

nS HIGH 50.000 % ;

 6 items analyzed, 1 timing error detected.

 Minimum period is  21.484ns.

--------------------------------------------------------------------------------
```

PERIOD Path with Phase

This is similar to the PERIOD constraint (without PHASE). The difference for this path is the source and destination clock. The destination clock defines which PERIOD constraint the path uses. Because the destination clock is the rclk_90, this path is in the TS_rclk90_dcm PERIOD and not the TS_rclk PERIOD constraint.

Notice the Requirement is now 2.5 ns and not 10 ns. This is the amount of time between the source clock (rising at 0ns) and the destination clock (rising at 2.5 ns).

Because the slack is negative, this path fails the constraint. In the Hierarchical Report Browser, this failing path is displayed in red.

```
Example:

--------------------------------------------------------------------------------
Slack:                  -2.871ns (requirement - (data path - clock skew
+ uncertainty))

  Source:               rd_addr[1] (FF)

  Destination:          ffl_reg (FF)


  Requirement:          2.500ns

  Data Path Delay:      5.224ns (Levels of Logic = 2)

  Clock Skew:      -0.147ns

  Source Clock:         rclk rising at 0.000ns

  Destination Clock:    rclk_90 rising at 2.500ns

  Clock Uncertainty:    0.000ns

  Data Path: rd_addr[1] to ffl_reg

  Location            Delay type         Delay(ns)  Logical Resource(s)

-------------------------------------------------  -------------------

    SLICE_X4Y19.XQ       Tcko                  0.568   rd_addr[1]

    SLICE_X2Y9.F3        net (fanout=40)       1.700   rd_addr[1]

    SLICE_X2Y9.X         Tilo                  0.439
full_st_i_0.G_4.G_4.G_3_10

    SLICE_X2Y11.F2       net (fanout=1)        0.459   G_3_10

    SLICE_X2Y11.X        Tilo                  0.439
full_st_i_0.G_4.G_4.G_4

    K4.O1                net (fanout=3)        1.230   G_4

    K4.OTCLK1            Tioock                0.389   ffl_reg
```

```
--------------------------------------------------------------------------------

    Total                                          5.224ns (1.835ns logic,
3.389ns route)

                                                  (35.1% logic, 64.9% route)

--------------------------------------------------------------------------------
```

Minimum Period Statistics

The Timing takes into account paths that are in a FROM:TO constraints but the minimum period value does not account for the extra time allowed by multi-cycle constraints.

An example of how the Minimum Period Statistics are calculated. This number is calculated assuming all paths are single cycle.

```
Example:
--------------------------------------------------------------------------------
Design statistics:
Minimum period:  30.008ns (Maximum frequency:  33.324MHz)
Maximum combinational path delay:  42.187ns
Maximum path delay from/to any node:  31.026ns
Minimum input arrival time before clock:  12.680ns
Maximum output required time before clock:  43.970ns
--------------------------------------------------------------------------------
```

Halting TRACE

To halt TRACE, enter **Ctrl+C** (on a workstation) or Ctrl-BREAK (on a PC). On a workstation, make sure that when you enter **Ctrl+C**, the active window is the window from which you invoked TRACE. The program prompts you to confirm the interrupt. Some files may be left when TRACE is halted (for example, a TRACE report file or a physical constraints file), but these files may be discarded because they represent an incomplete operation.

Chapter 13

Speedprint

Speedprint is compatible with the following families.

- Virtex™, Virtex™-E
- Virtex™-II
- Virtex™-II Pro, Virtex™-II Pro X
- Virtex™-4
- Virtex™-5 LX
- Spartan™-II, Spartan™-IIE
- Spartan™-3, Spartan™-3E, Spartan™-3L

This chapter contains the following sections.

Speedprint Overview

The Speedprint program lists block delays for a device's speed grade. This program supplements data sheets, but does not replace them.

The following figure shows the Speedprint design flow:

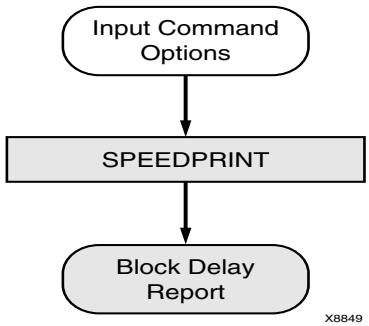


Figure 13-1: **Speedprint**

Speedprint Syntax

Use the following syntax to run speedprint:

speedprint [*options*] *device_name*

options can be any number of the Speedprint options listed in "Speedprint Options". They do not need to be listed in any particular order. Separate multiple options with spaces.

Speedprint Options

This section describes the options to the Speedprint command.

–intstyle (Integration Style)

-intstyle {**ise** | **xflow** | **silent**}

The –intstyle command line option specifies program invocation context. By default, the program is run as a standalone application.

-intstyle ise

Indicates that the program is being run as part of an integrated design environment.

-intstyle xflow

Indicates that the program is being run as part of a batch flow.

-intstyle silent

Indicates that only error messages and warnings will be displayed to the screen.

Note: The -intstyle option is automatically invoked when running in an integrated environment, such as Project Navigator or XFLOW.

–min (Display Minimum Speed Data)

The –min option displays minimum speed data for a device. This option overrides the –s option if both are used.

–s (Speed Grade)

–s [*speed_grade*]

The –s option with a speed_grade argument (for example, -4) displays data for the specified speed grade. If the –s option is omitted, delay data for the default, which is the fastest speed grade, is displayed.

–t (Specify Temperature)

–t *temperature*

The –t option specifies the operating die temperature in degrees Celsius. If this option is omitted, the worst-case temperature is used.

–v (Specify Voltage)

-v *voltage*

The –v option specifies the operating voltage of the device in volts. If this option is omitted, the worst-case voltage is used.

Speedprint Example Commands

The following table describes some example commands:

Command	Description
speedprint	Prints usage message
speedprint 2v80	Uses the default speed grade
speedprint –s -52v80 speedprint -s 5 2v80	Both displays block delays for speed grade -5
speedprint –2v50e -v 1.9 -t 40	Uses default speed grade at 1.9 volts and 40 degrees C
speedprint v50e -min	Displays data for the minimum speed grade

Speedprint Example Reports

Following is a portion of a speed grade report for a Virtex device. The following command generates the displayed report:

```
speedprint v100e
```

Note the new section at the end of the report. This section provides adjustments for the I/O values in the speed grade report according to the I/O standard you are using. These adjustments model the delay reporting in the data sheet. To use these numbers, add the adjustment for the standard you are using to the delays reported for the IOBs.

The default speed grade, temperature, and voltage settings are described at the beginning of the file.

```
Block delay report for a: xv100e
              Speed grade is: -8
Version id for speed file is: PRELIMINARY 1.60 2001-06-06 xilinx

This speed file supports voltage adjustments over the range of 1.700000
to 1.900000 volts.
Temperature adjustments are supported over the junction temperature
range of -40.000000 to 85.000000 degrees Celsius
Derated delay values for operating conditions within these limits are
available using this speed file.

This report prepared for default temperature and voltage.

Note - this report is intended to present the effect of different
speedgrades and voltage/temperature adjustments on block delays, for
specific situations use the timing analyzer report instead.
```

```
Delays are reported in picoseconds, where a range of delays is given
they represent the fastest and slowest paths reported under that name.

When a block is placed in a site normally used for another type of
block, a IOB placed in a Clock IOB site for example, small variations
in delay may occur which are not included in this report

External Setup and Hold requirements for global clocks
Tphf      -400          Tphfd          0          Tpsf         1500
Tpsfd     1800


Delays for a BLOCKRAM
Tback        831          Tbcka          0          Tbckd          0
Tbcke      -1097          Tbcko       2453          Tbckr       -961
Tbckw       -866          Tbdck        831          Tbeck       1928
Tbpwh       1164          Tbpwl       1164          Tbrck       1792
Tbwck       1697          Tgsrq       7531          Trpw       10100


Delays for a IOB
Tch     1116    Tcl     1116    Tgsrq   7531
Tgts    4050    Tiockice  1     Tiockiq 330 -  337
Tiockisr-482    Tiocko  -573    Tiockoce 1
Tiockon2736 - 2743Tiockosr-525  Tiockp  2335 - 2342
Tiockt  -238    Tiocktce-50     Tiocktsr-481
Tioiceck  546   Tioickp -985    Tioickpd  -2501
Tiooceck  546   Tioock  845     Tioolp    2872
Tioop     2473  Tiopi   744     Tiopick   1258
Tiopickd  2772  Tiopid  944     Tiopli    1385
.
.
.
.

I/O numbers in this report should be adjusted according to the I/O
standard being used. The adjustments are as follows:
                                        Input          Output
 Standard Name      Slew    Drive    Adjustment     Adjustment
 =============      ====    =====    ==========     ==========
 LVTTL               2      FAST            0          13001
 LVTTL               4      FAST            0           5201
 LVTTL               6      FAST            0           3001
 LVTTL               8      FAST            0            902
 LVTTL               12     FAST            0              0
 LVTTL               16     FAST            0            -50
 LVTTL               24     FAST            0           -200
 LVTTL               2      SLOW            0          14601
 LVTTL               4      SLOW            0           7401
 LVTTL               6      SLOW            0           4701
 LVTTL               8      SLOW            0           2902
.
.
.
.
```

Chapter 14

BitGen

BitGen is compatible with the following families:

- Virtex™, Virtex™-E
- Virtex™-II
- Virtex™-II Pro, Virtex™-II Pro X
- Virtex™-4
- Virtex™-5 LX
- Spartan™-II, Spartan™-IIE
- Spartan™-3, Spartan™-3E, Spartan™-3L

This chapter contains the following sections:

BitGen Overview

BitGen produces a bitstream for Xilinx device configuration. After the design is completely routed, it is necessary to configure the device so that it can execute the desired function. This is done using files generated by BitGen, the Xilinx bitstream generation program. BitGen takes a fully routed NCD (native circuit description) file as input and produces a configuration bitstream—a binary file with a .bit extension.

The BIT file contains all of the configuration information from the NCD file that defines the internal logic and interconnections of the FPGA, plus device-specific information from other files associated with the target device. The binary data in the BIT file is then downloaded into the FPGAs memory cells, or it is used to create a PROM file (see Chapter 16, "PROMGen").

The following figure shows the BitGen input and output files:

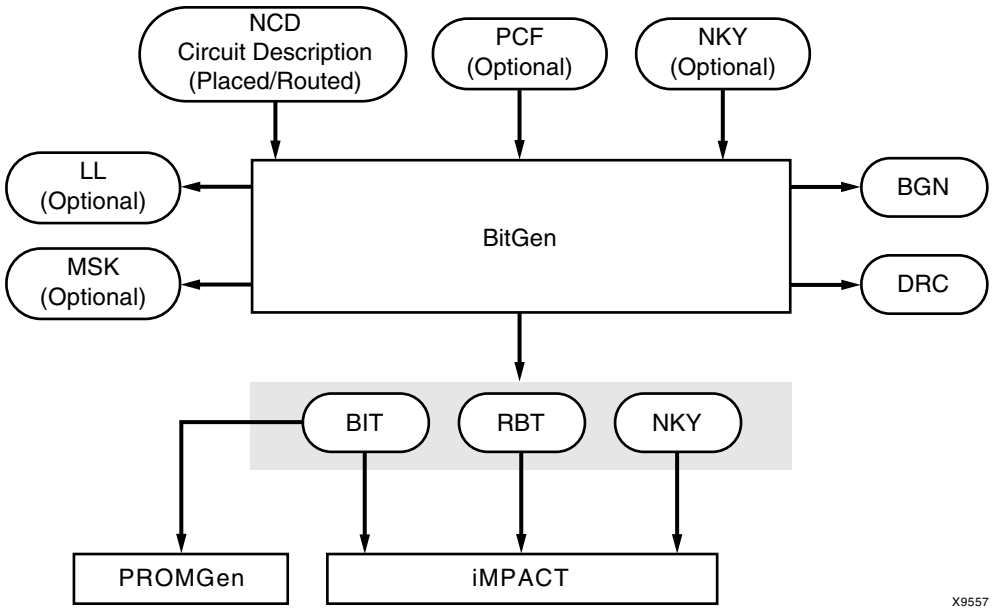


Figure 14-1: **BitGen input and output files**

BitGen Syntax

The following syntax creates a bitstream from your NCD file:

```
bitgen [options] infile[.ncd] [outfile] [pcf_file.pcf]
```

options is one or more of the options listed in "BitGen Options".

infile is the name of the NCD design for which you want to create the bitstream. You may specify only one design file, and it must be the first file specified on the command line.

Note: You do not have to use an extension. If you do not use an extension, then .ncd is assumed. If you do use an extension, then the extension must be .ncd.

outfile is the name of the output file. If you do not specify an output file name, BitGen creates a .bit file in your input file directory. If you specify any of the following options, the corresponding file is created in addition to the .bit file. If you do not specify an extension, BitGen appends the correct one for the specified option.

Option	Output File
–l	*outfile_name*.ll
–m	*outfile_name*.msk
–b	*outfile_name*.rbt

A report file containing all BitGen output is automatically created under the same directory as the output file. The report file has the same root name as the output file and a .bgn extension.

pcf_file is the name of a physical constraints file. BitGen uses this file to interpret CONFIG constraints, which control bitstream options. These CONFIG constraints override default behavior and can be overridden by configuration options. See "–g (Set Configuration)." BitGen automatically reads the .pcf file by default. If the PCF is the second file specified on the command line, it must have a .pcf extension. If it is the third file specified, the extension is optional; .pcf is assumed. If a .pcf file name is specified, it must exist; otherwise, the input design name with a .pcf extension is assumed.

Type the following syntax to see a complete list of BitGen command line options and supported devices:

```
bitgen –h
```

BitGen Input Files

Input to BitGen comprises the following files:

- NCD file—a physical description of the design mapped, placed and routed in the target device. The NCD file must be fully routed.
- PCF—an optional user-modifiable ASCII Physical Constraints File.
- NKY—an optional encryption key file.

Note: For more information on encryption, see the following web site: http://www.xilinx.com/products.

BitGen Output Files

Output from BitGen comprises the following files:

Table 14-1: **BitGen Output Files**

Output File Type	Output File Description
.bgn	Contains log information for the BitGen run, including command line options, errors, and warnings. Always produced.
.bin	A binary file that contains only configuration data. The .bin has no header like the .bit file. Produced when –g Binary:Yes is specified.
.bit	A binary file that contains proprietary header information as well as configuration data. Meant for input to other Xilinx tools, such as PROMGen and iMPACT. Always produced unless the -j option is specified.
.drc	A design rule check (DRC) file for the design. Contains log information or Design Rules Checker, including errors and warnings. Always produced unless the -d option is specified.
.isc	Contains the configuration data in IEEE1532 format. Produced when -g IEEE:1532:Yes is specified.
.ll	An ASCIII file that contains information on each of the nodes in the design that can be captured for readback. The file contains the absolute bit position in the readback stream, frame address, frame offset, logic resource used, and name of the component in the design. Produced when the -l option is specified.

Table 14-1: **BitGen Output Files**

Output File Type	Output File Description
.msd	An ASCII file that contains only mask information for verification, including pad words and frames. No commands are included. Produced when -g Readback is specified.
.msk	A binary file that contains the same configuration commands as a .bit file, but has mask data where the configuration data is. This data should NOT be used to configure the device. If a mask bit is 0, that bit should be verified against the bit stream data. If a mask bit is 1, that bit should not be verified. Produced when the -m option is specified.
.nky	An ASCII file that contains key information for Virtex-II devices when encryption is desired. This file is used as an input to iMPACT to program the keys. Produced when -g Encrypt:Yes is specified.
<outname>_key.isc	Contains the data for programming the encryption keys in IEEE 1532 format. Produced when -g IEEE 1532:Yes and -g Encrypt:Yes are set.
.rba	An ASCII file that contains readback commands, rather than configuration commands, and expected readback data where the configuration data would normally be. To produce the .rba file, the –b option must be used when –g Readback is specified.
.rbb	The same as the .rba file, but it is a binary file. Produced when –g Readback is specified.
.rbd	An ASCII file that contains only expected readback data, including pad words and frames. No commands are included. Produced when -g Readback is specified.
.rbt	An ASCII version of the bit file. Produced when the -b option is specified.

Note: For more information on encryption, see the Answers Database at the following web site: http://www.xilinx.com/support.

BitGen Options

Following is a description of the command line options and how they affect the behavior of BitGen.

Note: For a complete description of the Xilinx Development System command line syntax, see "Command Line Syntax" in Chapter 1.

–b (Create Rawbits File)

Create a *rawbits* (*file_name*.rbt) file. If the –g Readback option is specified in combination with the –b option, an ASCII readback command file (*file_name*.rba) is also generated.

The rawbits file consists of ASCII ones and zeros representing the data in the bitstream file. If you are using a microprocessor to configure a single FPGA, you can include the rawbits file in the source code as a text file to represent the configuration data. The sequence of characters in the rawbits file is the same as the sequence of bits written into the FPGA.

–bd (Update Block Rams)

```
-bd file_name
```

The –bd option updates the bitstream with the block ram content from the specified ELF or MEM file. See Chapter 24, "Data2MEM" for more information.

–d (Do Not Run DRC)

Do not run DRC (design rule check). Without the –d option, BitGen runs a DRC and saves the DRC results in two output files: the BitGen report file (*file_name*.bgn) and the DRC file (*file_name*.drc). If you enter the –d option, no DRC information appears in the report file and no DRC file is produced.

Running DRC before a bitstream is produced detects any errors that could cause the FPGA to malfunction. If DRC does not detect any errors, BitGen produces a bitstream file (unless you use the –j option described in "–j (No BIT File)").

–f (Execute Commands File)

–f *command_file*

The –f option executes the command line arguments in the specified *command_file*. For more information on the –f option, see "–f (Execute Commands File)" in Chapter 1.

–g (Set Configuration)

The –g option specifies the startup timing and other bitstream options for Xilinx FPGAs. The debug bitstream can only be used for master and slave serial configurations. It is not valid for Boundary Scan or Slave Parallel/Select MAP. The settings for the –g option depend on the architecture of the design. These settings are described in the following section:

–g (Set Configuration—Virtex/-E/-II/-II Pro/-4 and Spartan-II/-IIE/-3/-3E)

The –g option has sub-options that represent settings you use to set the configuration for a Virtex/-E/-II/-II Pro or Spartan-II/-IIE/3 design. These options have the following syntax:

bitgen -g *option*:*setting design*.**ncd** *design*.***bit*** *design*.***pcf***

For example, to enable Readback, use the following syntax:

```
bitgen -g Readback
```

The following sections describe the options and settings for the –g option. Each –g option is listed with supported architectures, settings, and defaults.

ActivateGCLK

Allows any partial bitstream for a reconfigurable area to have its registered elements wired to the correct clock domain. Clock domains must be minimally defined in the NCD.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	No, Yes
Default:	No

ActiveReconfig

Prevents the assertions of GHIGH and GSR during configuration. This is required for the active partial reconfiguration enhancement features.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	No, Yes
Default:	No

Binary

Creates a binary file with programming data only. Use this option to extract and view programming data. Any changes to the header will not affect the extraction process.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	No, Yes
Default:	No

CclkPin

Adds an internal pull-up to the Cclk pin. The Pullnone setting disables the pullup.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	Pullnone, Pullup
Default:	Pullup

Compress

This option uses the multiple frame write feature in the bitstream to reduce the size of the bitstream, not just the .bit file. Using the Compress option does not guarantee that the size of the bitstream will shrink. Compression is enabled by setting the BitGen option **`-g compress`**; compression is disabled by not setting it.

Note that the partial bit files generated with the BitGen **`-r`** setting automatically make use of the multiple frame write feature, and are *compressed* bitstreams.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Spartan-II, Spartan-IIE, Virtex-4, Spartan-3
Settings:	None
Default:	Off

ConfigRate

Virtex/-E/-II/-II Pro and Spartan-II/-IIE/-3 use an internal oscillator to generate the configuration clock, CCLK, when configuring in a master mode. Use the configuration rate option to select the rate for this clock.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan 3, Spartan-3E
Settings	4, 5, 7, 8, 9, 10, 13, 15, 20, 26, 30, 34, 41, 45, 51, 55, 60
Default:	4
Settings for Spartan-3/-3E	6, 3, 12, 25, 50, 100 (default is 6)
Default for Spartan-3:	6

Note: For a list of specific architecture settings, use the **`bitgen -h [architecture]`** command.The default value may vary by architecture.

CRC

The CRC option controls the generation of a Cyclic Redundancy Check value in the bitstream. When enabled, a unique CRC value is calculated based on bitstream contents. If the calculated CRC value does not match the CRC value in the bitstream, the device will fail to configure. When CRC is disabled a constant value is inserted in the bitstream in place of the CRC and the device will not calculate a CRC.

Architectures:	Virtex-II, Virtex-II Pro, Virtex-4, Spartan-3, Spartan-3E
Settings:	Disable, Enable
Default:	Enable

DCIUpdateMode

This option controls how often the Digitally Controlled Impedance circuit attempts to update the impedance match for DCI IOSTANDARDs. This option is preferable to the FreezeDCI option because it has no effect on bitstream size and can be used with Encrypted bitstreams. The setting DCIUpdateMode:Quiet supersedes the setting FreezeDCI:Yes.

Architectures:	Virtex-II Pro, Virtex-4, Spartan-3, Spartan-3E
Settings:	As required, continuous, quiet
Default:	As required

DCMShutdown

When DCMShutdown is enabled, the digital clock manager (DCM) resets if the SHUTDOWN and AGHIGH commands are loaded into the configuration logic.

Architectures:	Virtex-II, Virtex-II Pro, Virtex-4, Spartan-3, Spartan-3E
Settings:	Disable, Enable
Default:	Disable

DebugBitstream

If the device does not configure correctly, you can debug the bitstream using the DebugBitstream option. A debug bitstream is significantly larger than a standard bitstream. The values allowed for the DebugBitstream option are No and Yes.

Note: Use this option only if your device is configured to use slave or master serial mode.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Spartan-II, Spartan-IIE, Virtex-4, Spartan-3, Spartan-3E
Values:	No, Yes

In addition to a standard bitstream, a debug bitstream offers the following features:

- Writes 32 0s to the LOUT register after the synchronization word
- Loads each frame individually
- Performs a cyclical redundancy check (CRC) after each frame
- Writes the frame address to the LOUT register after each frame

DisableBandgap

Disables bandgap generator for DCMs to save power.

Architectures:	Virtex-II and Virtex-II Pro, Virtex-4,
Settings:	No, Yes
Default:	No

DONE_cycle

Selects the Startup phase that activates the FPGA Done signal. Done is delayed when DonePipe=Yes.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	1, 2, 3, 4, 5, 6
Default:	4

DonePin

Adds an internal pull-up to the DONE Pin pin. The Pullnone setting disables the pullup.

Use this option only if you are planning to connect an external pull-up resistor to this pin. The internal pull-up resistor is automatically connected if you do not use this option.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	Pullup, Pullnone
Default:	Pullup

DonePipe

This option is intended for use with FPGAs being set up in a high-speed daisy chain configuration.When set to Yes, the FPGA waits on the CFG_DONE (DONE) pin to go High and then waits for the first clock edge before moving to the Done state.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	No, Yes
Default:	No

DriveDone

This option actively drives the DONE Pin high as opposed to using a pullup.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	No, Yes
Default:	No

Encrypt

Encrypts the bitstream.

Architectures:	Virtex-II, Virtex-II Pro, Virtex-4,
Settings:	No, Yes
Default:	No

Note: For more information on encryption, see the following web site:
http://www.xilinx.com/products

Gclkdel0, Gclkdel1, Gclkdel2, Gclkdel3

Use these options to add delays to the global clocks. *Do not use these options unless instructed to do so by Xilinx*.

Architectures:	Virtex/-E/, Spartan-II/-IIE
Settings:	11111, *binary string*
Default:	11111

GSR_cycle

Selects the Startup phase that releases the internal set-reset to the latches, flip-flops, and BRAM output latches. The Done setting releases GSR when the DoneIn signal is High. DoneIn is either the value of the Done pin or a delayed version if DonePipe=Yes.

Architectures:	Virtex/-E, Spartan-II/-IIE
Settings:	Done, 1, 2, 3, 4, 5, 6, Keep
Default:	6

Keep should only be used when partial reconfiguration is going to be implemented. Keep prevents the configuration state machine from asserting control signals that could cause the loss of data.

GWE_cycle

Selects the Startup phase that asserts the internal write enable to flip-flops, LUT RAMs, and shift registers. It also enables the BRAMS. Before the Startup phase both BRAM writing and reading are disabled.The Done setting asserts GWE when the DoneIn signal is High. DoneIn is either the value of the Done pin or a delayed version if DonePipe=Yes. The Keep setting is used to keep the current value of the GWE signal.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	1, 2, 3, 4, 5, 6, Done, Keep
Default:	6

GTS_cycle

Selects the Startup phase that releases the internal 3-state control to the I/O buffers. The Done setting releases GTS when the DoneIn signal is High. DoneIn is either the value of the Done pin or a delayed version if DonePipe=Yes.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	Done, 1, 2, 3, 4, 5, 6, Keep
Default:	5

HswapenPin

Adds a pull-up, pull-down, or neither to the HSWAP_EN pin. The Pullnone option shows there is no connection to either the pull-up or the pull-down.

Architectures:	Virtex-II, Virtex-4, Spartan-3, Spartan-3E
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

Key0, Key1, Key2, Key3, Key4, Key5

Sets key*x* for bitstream encryption. The pick option causes BitGen to select a random number for the value.

Architectures:	Virtex-II, Virtex-II Pro, Virtex-4
Settings:	Pick, *hex_string*
Default:	Pick

Note: For more information on encryption, see the following web site: http://www.xilinx.com/products.

KeyFile

Specifies the name of the input encryption file.

Architectures:	Virtex-II, Virtex-II Pro, Virtex-4,
Settings:	*string*

Keyseq0, Keyseq1, Keyseq2, Keyseq3, Keyseq4, Keyseq5

Sets the key sequence for key*x*. The settings are equal to the following:

- S=single
- F=first
- M=middle
- L=last

Architectures:	Virtex-II, Virtex-II Pro, Virtex-4,
Settings:	S, F, M, L
Default:	S

LCK_cycle

Selects the Startup phase to wait until DLLs/DCMs lock. If NoWait is selected, the Startup sequence does not wait for DLLs/DCMs.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	0,1, 2, 3, 4, 5, 6, NoWait
Default:	NoWait

M0Pin

Adds an internal pull-up, pull-down or neither to the M0 pin. The following settings are available. The default is PullUp. Select Pullnone to disable both the pull-up resistor and pull-down resistor on the M0 pin.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

M1Pin

Adds an internal pull-up, pull-down or neither to the M1 pin. The following settings are available. The default is PullUp.

Select Pullnone to disable both the pull-up resistor and pull-down resistor on the M1 pin.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

M2Pin

Adds an internal pull-up, pull-down or neither to the M2 pin. The default is PullUp. Select Pullnone to disable both the pull-up resistor and pull-down resistor on the M2 pin.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

Match_cycle

Specifies a stall in the Startup cycle until digitally controlled impedance (DCI) match signals are asserted.

Architectures:	Virtex-II, Virtex-II Pro, Virtex-4, Spartan-3, Spartan-3E
Settings:	Auto, NoWait, 0, 1, 2, 3, 4, 5, 6
Default:	NoWait

Note: When the Auto setting is specified, BitGen searches the design for any DCI I/O standards. If DCI standards exist, BitGen will use the Match_cycle:2 setting, otherwise it will use the Match_cycle:NoWait setting.

PartialGCLK

Adds the center global clock column frames into the list of frames to write out in a partial bitstream. This option is equivalent to the PartialMask0:1 option.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Default:	<Not Specified> - no partial masks in use

PartialMask0, PartialMask1, PartialMask2

Generates a bitstream comprised of only the major addresses of block type *<0, 1, or 2>* that have enabled value in the mask. The block type is all non-block ram initialization data frames in the applicable device and its derivatives. The mask is a hex value.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	All columns enabled, major address mask
Default:	<Not Specified> - no partial masks in use

PartialLeft

Adds the left side frames of the device into the list of frames to write out in a partial bitstream. This includes CLB, IOB, and BRAM columns. It does not include the center global clock column.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E

PartialRight

Adds the right side frames of the device into the list of frames to write out in a partial bitstream. This includes CLB, IOB, and BRAM columns. It does not include the center global clock column.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E

Persist

This option is needed for Readback and Partial Reconfiguration using the SelectMAP configuration pins. If Persist is set to Yes, the pins used for SelectMAP mode are prohibited for use as user I/O. Refer to the datasheet for a description of SelectMAP mode and the associated pins.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	No, Yes
Default:	No

PowerdownPin

Puts the pin into “sleep” mode by specifying whether or not the internal pullup on the pin is enabled.

Architectures:	Virtex-II, Virtex-II Pro, Virtex-4,
Settings:	Pullup, Pullnone
Default:	Pullup

ProgPin

Adds an internal pull-up to the ProgPin pin. The Pullnone setting -disables the pullup. The pull-up affects the pin after configuration.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	Pullup, Pullnone
Default:	Pullup

ReadBack

This option allows you to perform the Readback function by creating the necessary readback files.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, and Spartan-3E

When specifying the –g Readback option, the .rbb, .rbd, and .msd files are created.

If the –b option is used in conjunction with the –g Readback option, an ASCII readback command file (*file_name*.rba) is also generated.

Security

Selecting Level1 disables Readback. Selecting Level2 disables Readback and Partial Reconfiguration.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	None, Level1, Level2
Default:	None

SEURepair

This option supports single event upset repair by writing bitstreams a single frame at a time, rather than in one packet. Frame Address Register (FAR) headers are written to sequentially. First the FRAME address is written to, followed by the FRAME data.

Architectures:	Virtex-II, Virtex-II Pro, Virtex-4, Spartan-3, Spartan-3E
Settings:	No, Yes
Default:	No

StartCBC

Sets the starting cipher block chaining (CBC) value. The pick option causes BitGen to select a random number for the value.

Architectures:	Virtex-II, Virtex-II Pro, Virtex-4,
Settings:	Pick, *hex_string*
Default:	Pick

StartKey

Sets the starting key number.

Architectures:	Virtex-II, Virtex-II Pro
Settings:	0, 3
Default:	0

StartupClk

The startup sequence following the configuration of a device can be synchronized to either Cclk, a User Clock, or the JTAG Clock. The default is Cclk.

- Cclk

  Enter Cclk to synchronize to an internal clock provided in the FPGA device.

- JTAG Clock

  Enter JtagClk to synchronize to the clock provided by JTAG. This clock sequences the TAP controller which provides the control logic for JTAG.

- UserClk

  Enter UserClk to synchronize to a user-defined signal connected to the CLK pin of the STARTUP symbol.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan 3, and Spartan- 3E
Settings:	Cclk (pin—see Note), UserClk (user-supplied), JtagCLK
Default:	Cclk

Note: In modes where Cclk is an output, the pin is driven by an internal oscillator.

TckPin

Adds a pull-up, a pull-down or neither to the TCK pin, the JTAG test clock. Selecting one setting enables it and disables the others. The Pullnone setting shows there is no connection to either the pull-up or the pull-down.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

TdiPin

Adds a pull-up, a pull-down, or neither to the TDI pin, the serial data input to all JTAG instructions and JTAG registers. Selecting one setting enables it and disables the others. The Pullnone setting shows there is no connection to either the pull-up or the pull-down.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

TdoPin

Adds a pull-up, a pull-down, or neither to the TdoPin pin, the serial data output for all JTAG instruction and data registers. Selecting one setting enables it and disables the others. The Pullnone setting shows there is no connection to either the pull-up or the pull-down.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

TmsPin

Adds a pull-up, pull-down, or neither to the TMS pin, the mode input signal to the TAP controller. The TAP controller provides the control logic for JTAG. Selecting one setting enables it and disables the others. The Pullnone setting shows there is no connection to either the pull-up or the pull-down

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

UnusedPin

Adds a pull-up, a pull-down, or neither to the unused device pins and the serial data output (TDO) for all JTAG instruction and data registers. Selecting one setting enables it and disables the others. The Pullnone setting shows there is no connection to either the pull-up or the pull-down.

The following settings are available. The default is Pulldown.

Architectures:	Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3, Spartan-3E
Settings:	Pullup, Pulldown, Pullnone
Default:	Pulldown

UserID

You can enter up to an 8-digit hexadecimal code in the User ID register. You can use the register to identify implementation revisions.

Architectures:	Virtex-4, Spartan-3, Spartan-3E
Settings:	0xFFFFFFFF, [*hex string]*
Default:	0xFFFFFFFF

–intstyle (Integration Style)

```
-intstyle {ise | xflow | silent}
```

The –intstyle option reduces screen output based on the integration style you are running. When using the –intstyle option, one of three modes must be specified: *ise*, *xflow*, or *silent*. The mode sets the way information is displayed in the following ways:

`-intstyle ise`

This mode indicates the program is being run as part of an integrated design environment.

`-intstyle xflow`

This mode indicates the program is being run as part of an integrated batch flow.

`-intstyle silent`

This mode limits screen output to warning and error messages only.

Note: The -intstyle option is automatically invoked when running in an integrated environment, such as Project Navigator or XFLOW.

–j (No BIT File)

Do not create a bitstream file (.bit file). This option is used when you want to generate a report without producing a bitstream. For example, if you wanted to run DRC without producing a bitstream file, you would use the -j option.

Note: The .msk or .rbt files may still be created.

–l (Create a Logic Allocation File)

This option creates an ASCII logic allocation file (*design*.ll) for the selected design. The logic allocation file shows the bitstream position of latches, flip-flops, IOB inputs and outputs, and the bitstream position of LUT programming and Block RAMs.

In some applications, you may want to observe the contents of the FPGA internal registers at different times. The file created by the –l option helps you identify which bits in the current bitstream represent outputs of flip-flops and latches. Bits are referenced by frame and bit number within the frame.

The iMPACT tool uses the design.ll file to locate signal values inside a readback bitstream.

–m (Generate a Mask File)

Creates a mask file. This file determines which bits in the bitstream should be compared to readback data for verification purposes.

–r (Create a Partial Bit File)

–r *bit_file*

The –r option is used to create a partial bit file. It takes that bit file and compares it to the .ncd file given to bitgen. Instead of writing out a full bit file, it only writes out the part of the bit file that is different from the original bit file given.

–w (Overwrite Existing Output File)

Enables you to overwrite an existing BitGen output file. See "BitGen Output Files" for additional information.

Chapter 15

BSDLAnno

BSDLAnno is compatible with the following families:

- Virtex™, Virtex™-E
- Virtex™-II
- Virtex™-II Pro, Virtex™-II Pro X
- Virtex™-4
- Virtex™-5 LX
- Spartan™-II, Spartan™-IIE
- Spartan™-3, Spartan™-3E, Spartan™-3L
- CoolRunner™ XPLA3, CoolRunner™-II
- XC9500™, XC9500XL™, XC9500XV™

This chapter contains the following sections:

BSDLAnno Overview

The Boundary Scan Description Language (BSDLAnno) utility automatically modifies a BSDL file for post-configuration interconnect testing. BSDLAnno obtains the necessary design information from the routed .ncd file (FPGAs) or the design.pnx file (CPLDs), and generates a BSDL file that reflects the post-configuration boundary scan architecture of the device. The boundary scan architecture is changed when the device is configured because certain connections between the boundary scan registers and pad may change. These changes must be communicated to the boundary scan tester through a post-configuration BSDL file. If the changes to the boundary scan architecture are not reflected in the BSDL file, boundary scan tests may fail.

The Boundary Scan Description Language is defined by IEEE specification 1149.1 as "a common way of defining device boundary scan architecture." Xilinx provides both 1149.1 and 1532 BSDL files that describe pre-configuration boundary scan architecture.

For most Xilinx device families, the boundary scan architecture changes after the device is configured because the boundary scan registers sit behind the output buffer and the input sense amplifier:

```
BSCAN Register -> output buffer/input sense amp -> PAD
```

The hardware is arranged in this way so that the boundary scan logic operates at the I/O standard specified by the design. This allows boundary scan testing across the entire range of available I/O standards.

BSDLAnno Syntax

The following syntax creates a post-configuration BSDL file with BSDLAnno:

```
bsdlanno [options] infile outfile[.bsd]
```

options is one or more of the options listed in "BSDLAnno Options".

infile is the design source file for the specified design. For FPGA designs, the infile is a routed (post-PAR) NCD file. For CPLD designs, the infile is the *design*.pnx file.

outfile is the destination for the design-specific BSDL file with an optional .bsd extension. The length of the BSDL output filename, including the .bsd extension, cannot exceed 24 characters.

BSDLAnno Input Files

BSDLAnno requires two input files to generate a post-configuration BSDL file:

- Pre-configuration BSDL (.bsd) file that is automatically read from the Xilinx installation area.
- The routed .ncd file (FPGAs) or the .pnx file (CPLDs), which is specified as the infile.

BSDLAnno Output Files

The output from BSDLAnno is an ASCII (text) formatted BSDL file that has been modified to reflect signal direction (input/output/bidirectional), unused I/Os, and other design-specific boundary scan behavior.

BSDLAnno Options

This section provides information on the BSDLAnno command line options.

–s (Specify BSDL file)

```
-s [IEEE1149 | IEEE1532]
```

The –s option specifies the pre-configuration BSDL file to be annotated. IEEE1149 and IEEE1532 versions of the pre-configuration BSDL file are currently available.

Note: Most users require the IEEE1149 version.

–intstyle (Integration Style)

-intstyle {**ise** | **xflow** | **silent**}

The –intstyle option reduces screen output based on the integration style you are running. When using the –intstyle option, one of three modes must be specified: *ise*, *xflow*, or *silent*. The mode sets the way information is displayed in the following ways:

-intstyle ise

This mode indicates the program is being run as part of an integrated design environment.

-intstyle xflow

This mode indicates the program is being run as part of an integrated batch flow.

-intstyle silent

This mode limits screen output to warning and error messages only.

Note: The -intstyle option is automatically invoked when running in an integrated environment, such as Project Navigator or XFLOW.

BSDLAnno File Composition

Manufacturers of JTAG-compliant devices must provide BSDL files for those devices. BSDL files describe the boundary scan architecture of a JTAG-compliant device, and are written in a subset language of VHDL. The main parts of an IEEE1149 BSDL file follow, along with an explanation of how BSDLAnno modifies each section.

Entity Declaration

The entity declaration is a VHDL construct that is used to identify the name of the device that is described by the BSDL file.

For example (from the xcv50e_pq240.bsd file): entity XCV50E_PQ240 is *design_name*.[**ncd/pnx]**

BSDLAnno changes the entity declaration to avoid name collisions. The new entity declaration matches the design name in the input .ncd or .pnx file.

Generic Parameter

The generic parameter specifies which package is described by the BSDL file.

```
For example (from the xcv50e_pq240.bsd file):
generic (PHYSICAL_PIN_MAP : string := "PQ240" );
```

BSDLAnno does not modify the generic parameter.

Logical Port Description

The logical port description lists all I/Os on a device and states whether the pin is input, output, bidirectional, or unavailable for boundary scan. Pins configured as outputs are described as *inout* because the input boundary scan cell remains connected, even when the pin is used only as an output. Describing the output as *inout* reflects the actual boundary scan capability of the device and allows for greater test coverage.

Not all I/Os on the die are available (or bonded) in all packages. Unbonded I/Os are defined in the pre-configuration BSDL file as *linkage* bits.

```
For example (from the xcv50e_pq240.bsd file):
port (
CCLK_P179: inout bit;
DONE_P120: inout bit;
GCK0_P92: in bit;
GCK1_P89: in bit;
GCK2_P210: in bit;
GCK3_P213: in bit;
GND: linkage bit_vector (1 to 32);
INIT_P123: inout bit; -- PAD96
IO_P3: inout bit; -- PAD191
IO_P4: inout bit; -- PAD190
IO_P5: inout bit; -- PAD189
IO_P6: inout bit; -- PAD188
```

BSDLAnno modifies the logical port description to match the capabilities of the boundary scan circuitry after configuration. Modifications are made as follows:

- Dedicated pins (JTAG, mode, done, etc.) are not modified; they are left as *inout bit*.
- Pins defined as bidirectional are left as *inout bit*
- Pins defined as inputs are changed to *in bit*
- Pins defined as outputs are left as *inout bit*
- Unused pins are not modified
- The N-side of differential pairs is changed to *linkage bit*

Package Pin-Mapping

Package pin-mapping shows how the pads on the device die are wired to the pins on the device package.

```
For example (from the xcv50e_pq240.bsd file):
"CCLK_P179:P179," &
"DONE_P120:P120," &
"GCK0_P92:P92," &
"GCK1_P89:P89," &
"GCK2_P210:P210," &
"GCK3_P213:P213," &
"GND:(P1,P8,P14,P22,P29,P37,P45,P51,P59,P69," &
"P75,P83,P91,P98,P106,P112,P119,P129,P135,P143," &
```

```
"P151,P158,P166,P172,P182,P190,P196,P204,P211,P219," &
"P227,P233)," &
"INIT_P123:P123," &
"IO_P3:P3," &
"IO_P4:P4," &
"IO_P5:P5," &
"IO_P6:P6," &
```

BSDLAnno does not modify the package pin-mapping.

USE Statement

The USE statement calls VHDL packages that contain attributes, types, and constants that are referenced in the BSDL file.

```
For example (from the xcv50e_pq240.bsd file):
use STD_1149_1_1994.all;
```

BSDLAnno does not modify USE statements.

Scan Port Identification

The scan port identification identifies the following JTAG pins: TDI, TDO, TMS, TCK and TRST.

Note: TRST is an optional JTAG pin that is not used by Xilinx devices.

```
For example (from the xcv50e_pq240.bsd file):
attribute TAP_SCAN_IN of TDI : signal is true;
attribute TAP_SCAN_MODE of TMS : signal is true;
attribute TAP_SCAN_OUT of TDO : signal is true;
attribute TAP_SCAN_CLOCK of TCK : signal is (33.0e6, BOTH);
```

BSDLAnno does not modify the Scan Port Identification.

TAP Description

The TAP description provides additional information on the JTAG logic of a device. Included are the instruction register length, instruction opcodes, and device IDCODE. These characteristics are device-specific and may vary widely from device to device.

```
For example (from the xcv50e_pq240.bsd file):
attribute COMPLIANCE_PATTERNS of XCV50E_PQ240 : entity is
attribute INSTRUCTION_LENGTH of XCV50E_PQ240 : entity is 5;
attribute INSTRUCTION_OPCODE of XCV50E_PQ240 : entity is
attribute INSTRUCTION_CAPTURE of XCV50E_PQ240 : entity is "XXX01";
attribute IDCODE_REGISTER of XCV50E_PQ240 : entity is
```

BSDLAnno does not modify the TAP Description.

Boundary Register Description

The boundary register description gives the structure of the boundary scan cells on the device. Each pin on a device may have up to three boundary scan cells, with each cell consisting of a register and a latch. Boundary scan test vectors are loaded into or scanned from these registers.

```
For example (from the xcv50e_pq240.bsd file):
attribute BOUNDARY_REGISTER of XCV50E_PQ240 : entity is
-- cellnum (type, port, function, safe[, ccell, disval, disrslt])
" 0 (BC_1, *, controlr, 1)," &
" 1 (BC_1, IO_P184, output3, X, 0, 1, PULL0)," & -- PAD48
" 2 (BC_1, IO_P184, input, X)," & -- PAD48
```

Every IOB has three boundary scan registers associated with it: control, output, and input. BSDLAnno modifies the boundary register description as described in the "BSDL File Modifications for Single-Ended Pins" and "BSDL File Modifications for Differential Pins" sections.

BSDL File Modifications for Single-Ended Pins

If pin 57 has been configured as a single-ended tri-state output pin, no code modifications are required:

```
-- TRISTATE OUTPUT PIN (three state output with an input component)
" 9 (BC_1, *, controlr, 1)," &
" 10 (BC_1, PAD57, output3, X, 9, 1, Z)," &
" 11 (BC_1, PAD57, input, X)," &
```

If pin 57 is configured as a single-ended input, modify as follows:

```
-- PIN CONFIGURED AS AN INPUT
" 9 (BC_1, *, internal, 1)," &
" 10 (BC_1, *, internal, X)," &
" 11 (BC_1, PAD57, input, X)," &
```

If pin 57 is configured as a single-ended output, it is treated as a single-ended bidirectional pin:

```
-- PIN CONFIGURED AS AN OUTPUT
" 9 (BC_1, *, controlr, 1)," &
" 10 (BC_1, PAD57, output3, X, 9, 1, Z)," &
" 11 (BC_1, PAD57, input, X)," &
```

If pin 57 is unconfigured or not used in the design, do not modify:

```
-- PIN CONFIGURED AS "UNUSED"
" 9 (BC_1, *, controlr, 1)," &
" 10 (BC_1, PAD57, output3, X, 9, 1, PULL0)," &
" 11 (BC_1, PAD57, input, X)," &
```

Explanation:

The only modification that is made to single-ended pins is when the pin is configured as an input. In this case, the boundary scan logic is disconnected from the output driver and is unable to drive out on the pin. When a pin is configured as an output, the boundary scan input register remains connected to that pin. As a result, the boundary scan logic has the same capabilities as if the pin were configured as a bidirectional pin.

BSDL File Modifications for Differential Pins

If pin 57 is configured as a differential output, differential three-state output, or differential bidirectional pin, modify as follows:

```
" 9 (BC_1, *, controlr, 1)," &
" 10 (BC_1, PAD57, output3, X, 9, 1, Z)," &
" 11 (BC_1, PAD57, input, X)," &
```

If pin 57 is configured as a p-side differential input pin, modify as follows:

```
" 9 (BC_1, *, internal, 1)," &
" 10 (BC_1, *, internal, X)," &
" 11 (BC_1, PAD57, input, X)," &
```

If pin 57 is configured as an n-side differential pin (all types: input, output, 3-state output, and bidirectional), modify as follows:

```
" 9 (BC_1, *, internal, 1)," &
" 10 (BC_1, *, internal, X)," &
" 11 (BC_1, *, internal, X)," &
```

Explanation:

All interactions with differential pin pairs are handled by the boundary scan cells connected to the P-side pin. To capture the value on a differential pair, scan the P-side input register. To drive a value on a differential pair, shift the value into the P-side output register. The values in the N-side scan registers have no effect on that pin.

Most boundary scan devices use only three boundary scan registers for each differential pair. Most devices do not offer direct boundary scan control over each individual pin, but rather over the two pin pair. This makes sense when you consider that the two pins are transmitting only one bit of information - hence only one input, output, and control register is needed. Confusion arises over how differential pins are handled in Xilinx devices, because there are three boundary scan cells for each pin, or six registers for the differential pair. The N-side registers remain in the boundary scan register but are not connected to the pin in any way, which is why the N-side registers are listed as *internal* registers in the post-configuration BSDL file. The behavior of the N-side pin is controlled by the P-side boundary scan registers. For example, when a value is placed in the P-side output scan register and the output is enabled, the inverse value is driven onto the N-side pin by the output driver. This is independent from the Boundary Scan logic.

Modifications to the DESIGN_WARNING Section

BSDLAnno adds the following DESIGN_WARNING to the BSDL file:

```
"This BSDL file has been modified to reflect post-configuration"&
behavior by BSDLAnno. BSDLAnno does not modify the USER1,"&
USER2, or USERCODE registers. For details on the features and" &
limitations of BSDLAnno, please consult the Xilinx Development" &
System Reference Guide.";
```

Header Comments

BSDLAnno adds the following comments to the BSDL file header:

- BSDLAnno Post-Configuration File for design [*entity name*]
- BSDLAnno [*BSDLAnno version number*]

Boundary Scan Behavior in Xilinx Devices

BSDL files provided by Xilinx reflect the boundary scan behavior of an unconfigured device. After configuration, the boundary scan behavior of a device may change. I/O pins that were bidirectional before configuration may now be input-only. Boundary Scan test vectors are typically derived from BSDL files; therefore, if boundary scan tests are going to be performed on a configured Xilinx device, the BSDL file should be modified to reflect the configured boundary scan behavior of the device.

Whenever possible, boundary scan tests should be performed on an unconfigured Xilinx device. Unconfigured devices allow for better test coverage, because all I/Os are available for bidirectional scan vectors.

In most cases, boundary scan tests with Xilinx devices must be performed after FPGA configuration only under the following circumstances:

- When configuration cannot be prevented
- When differential signaling standards are used, unless the differential signals are located between Xilinx devices, in which case both devices can be tested before configuration. Each side of the differential pair will behave as a single-ended signal.

Chapter 16

PROMGen

PROMGen is compatible with the following families:

- Virtex™, Virtex™-E
- Virtex™-II
- Virtex™-II Pro, Virtex™-II Pro X
- Virtex™-4
- Virtex™-5 LX
- Spartan™-II, Spartan™-IIE
- Spartan™-3, Spartan™-3E, Spartan™-3L

This chapter contains the following sections:

PROMGen Overview

PROMGen formats a BitGen-generated configuration bitstream (BIT) file into a PROM format file. The PROM file contains configuration data for the FPGA device. PROMGen converts a BIT file into one of three PROM formats: MCS-86 (Intel), EXORMAX (Motorola), or TEKHEX (Tektronix). It can also generate a binary or hexadecimal file format.

The following figure shows the inputs and the possible outputs of the PROMGen program:

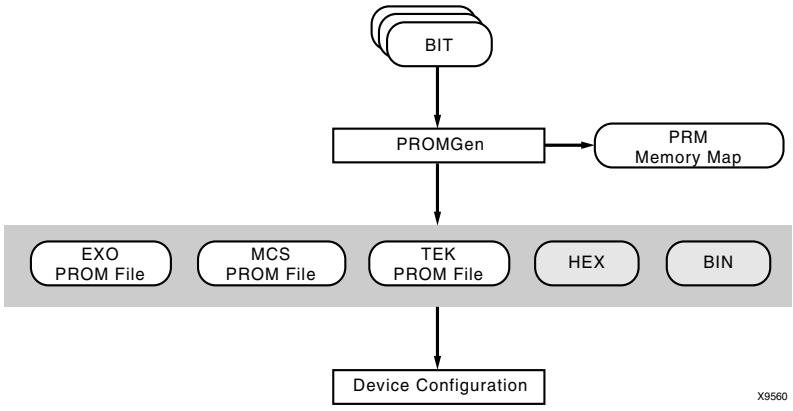


Figure 16-1: **PROMGen**

There are two functionally equivalent versions of PROMGen. There is a stand-alone version that you can access from an operating system prompt. There is also an interactive version, called the PROM formatting wizard that you can access from inside Project Navigator (see the *iMPACT online help*).

You can also use PROMGen to concatenate bitstream files to daisy-chain FPGAs.

Note: If the destination PROM is one of the Xilinx Serial PROMs, you are using a Xilinx PROM Programmer, and the FPGAs are not being daisy-chained, it is not necessary to make a PROM file.

PROMGen Syntax

To start PROMGen from the operating system prompt, use the following syntax:

promgen [*options*]

options can be any number of the options listed in "PROMGen Options". Separate multiple options with spaces.

PROMGen Input Files

The input to PROMGEN consists of one or more BIT and RBT files. BIT files contain configuration data for an FPGA design.

PROMGen Output Files

Output from PROMGEN consists of the following files:

- PROM files—The file or files containing the PROM configuration information. Depending on the PROM file format your PROM programmer uses, you can output a TEK, MCS, BIN, or EXO file. If you are using a microprocessor to configure your devices, you can output a HEX file, which contains a hexadecimal representation of the bitstream.
- PRM file—The PRM file is a PROM image file. It contains a memory map of the output PROM file. The file has a .prm extension.

PROMGen Options

This section describes the options that are available for the PROMGen command.

–b (Disable Bit Swapping—HEX Format Only)

This option only applies if the –p option specifies a HEX file for the output of PROMGen. By default (no –b option), bits in the HEX file are swapped compared to bits in the input BIT files. If you enter a –b option, the bits are not swapped. Bit swapping is described in "Bit Swapping in PROM Files".

–c (Checksum)

```
promgen -c
```

The –c option generates a checksum value appearing in the .prm file. This value should match the checksum in the prom programmer. Use this option to verify that correct data was programmed into the prom.

–d (Load Downward)

promgen –d *hexaddress0 filename filename . . .*

This option loads one or more BIT files from the starting address in a downward direction. Specifying several files after this option causes the files to be concatenated in a daisy chain. You can specify multiple –d options to load files at different addresses. You must specify this option immediately before the input bitstream file.

Here is the multiple file syntax.

promgen –d *hexaddress0 filename filename . . .*

Here is the multiple –d options syntax.

promgen –d *hexaddress1 filename* **-d** *hexaddress2 filename...*

–f (Execute Commands File)

–f *command_file*

The –f option executes the command line arguments in the specified *command_file*. For more information on the –f option, see "–f (Execute Commands File)" in Chapter 1.

–i (Select Initial Version)

-i *version*

The –i option is used to specify the initial version for a Xilinx multi-bank PROM.

–l (Disable Length Count)

```
promgen -l
```

The –l option disables the length counter in the FPGA bitstream. Use this option when chaining together bitstreams exceeding the 24 bit limit imposed by the length counter.

–n (Add BIT FIles)

–n *file1* [**.bit**] *file2* [**.bit**] . . .

This option loads one or more BIT files up or down from the next available address following the previous load. The first –n option *must* follow a –u or –d option because -n does not establish a direction. Files specified with this option are not daisy-chained to previous files. Files are loaded in the direction established by the nearest prior –u, –d, or –n option.

The following syntax shows how to specify multiple files. When you specify multiple files, PROMGen daisy-chains the files.

promgen –d *hexaddress file0* **–n** *file1 file2...*

The syntax for using multiple –n options follows. Using this method prevents the files from being daisy-chained.

promgen –d *hexaddress file0* **–n** *file1* **-n** *file2...*

–o (Output File Name)

–o *file1* [*.ext*] *file2* [*.ext*] . . .

This option specifies the output file name of a PROM if it is different from the default. If you do not specify an output file name, the PROM file has the same name as the first BIT file loaded.

ext is the extension for the applicable PROM format.

Multiple file names may be specified to split the information into multiple files. If only one name is supplied for split PROM files (by you or by default), the output PROM files are named *file*_#.ext, where *file* is the base name, # is 0, 1, etc., and *ext* is the extension for the applicable PROM format.

promgen –d *hexaddress file0* **–o** *filename*

–p (PROM Format)

-p {mcs | exo | tek | hex| bin| ufp| ieee1532}

This option sets the PROM format to MCS (Intel MCS86), EXO (Motorola EXORMAX), or TEK (Tektronix TEKHEX). The option may also produce a HEX file, which is a hexadecimal representation of the configuration bitstream used for microprocessor downloads. The default format is MCS.

The option may also produce a bin file, which is a binary representation of the configuration bitstream used for microprocessor downloads.

–r (Load PROM File)

–r *promfile*

This option reads an existing PROM file as input instead of a BIT file. All of the PROMGen output options may be used, so the –r option can be used for splitting an existing PROM file into multiple PROM files or for converting an existing PROM file to another format.

–s (PROM Size)

–s *promsize1 promsize2...*

This option sets the PROM size in kilobytes. The PROM size must be a power of 2. The default value is 64 kilobytes. The –s option must precede any –u, –d, or –n options.

Multiple *promsize* entries for the –s option indicates the PROM will be split into multiple PROM files.

Note: PROMGen PROM sizes are specified in bytes. See the –x option for more information.

–t (Template File)

–t *templatefile*.**pft**

The –t option specifies a template file for the user format PROM (UFP). If unspecified, the default file $XILINX/data/default.pft is used. If the UFP format is selected, the –t option is used to specify a control file.

–u (Load Upward)

–u *hexaddress0 filename1 filename2...*

This option loads one or more BIT files from the starting address in an upward direction. When you specify several files after this option, PROMGen concatenates the files in a daisy chain. You can load files at different addresses by specifying multiple –u options.

This option must be specified immediately before the input bitstream file.

–ver (Version)

–ver *[version] hexaddress filename1.bit filename2.bit . . .*

The –ver option loads .bit files from the specified hexaddress. Multiple .bit files daisychain to form a single PROM load. The daisychain is assigned to the specified version within the PROM.

Note: This option is only valid for Xilinx multi-bank PROMs.

–w (Overwrite Existing Output File)

```
promgen -w
```

The –w option overwrites an existing output file, and *must* be used if an output file exists. If this option is not used, PROMGen issues an error.

–x (Specify Xilinx PROM)

–x *xilinx_prom1 xilinx_prom2...*

The –x option specifies one or more Xilinx serial PROMs for which the PROM files are targeted. Use this option instead of the –s option if you know the Xilinx PROMs to use.

Multiple *xilinx_prom* entries for the –x option indicates the PROM will be split into multiple PROM files.

–z (Enable Compression)

-z *version*

The –z option enables compression for a Xilinx multi-bank PROM. All version will be compressed if one is not specified.

Bit Swapping in PROM Files

PROMGen produces a PROM file in which the bits within a byte are swapped compared to the bits in the input BIT file. Bit swapping (also called "bit mirroring") reverses the bits within each byte, as shown in the following figure:

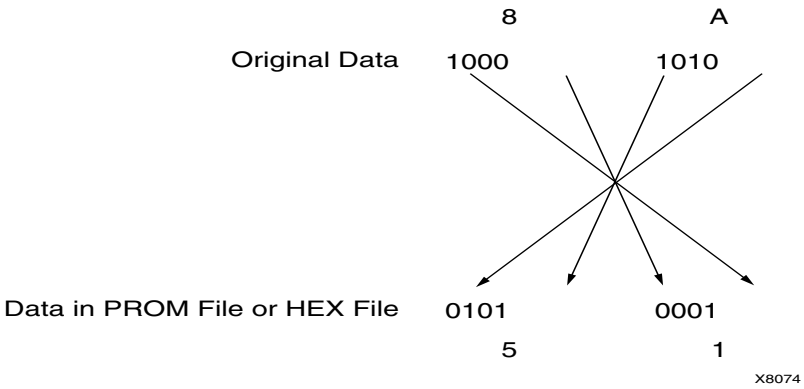


Figure 16-2: **Bit Swapping**

In a bitstream contained in a BIT file, the Least Significant Bit (LSB) is always on the left side of a byte. But when a PROM programmer or a microprocessor reads a data byte, it identifies the LSB on the right side of the byte. In order for the PROM programmer or microprocessor to read the bitstream correctly, the bits in each byte must first be swapped so they are read in the correct order.

In this release of the Xilinx Development System, the bits are swapped for all of the PROM formats: MCS, EXO, BIN, and TEK. For a HEX file output, bit swapping is on by default, but it can be turned off by entering a –b PROMGen option that is available only for HEX file format.

PROMGen Examples

To load the file test.bit up from address 0x0000 in MCS format, enter the following information at the command line:

```
promgen -u 0 test
```

To daisy-chain the files test1.bit and test2.bit up from address 0x0000 and the files test3.bit and test4.bit from address 0x4000 while using a 32K PROM and the Motorola EXORmax format, enter the following information at the command line:

```
promgen -s 32 -p exo -u 00 test1 test2 -u 4000 test3 test4
```

To load the file test.bit into the PROM programmer in a downward direction starting at address 0x400, using a Xilinx XC1718D PROM, enter the following information at the command line:

```
promgen -x xc1718d -u 0 test
```

To specify a PROM file name that is different from the default file name enter the following information at the command line:

```
promgen options filename -o newfilename
```

Chapter 17

IBISWriter

The IBISWriter program is compatible with the following families:

- Virtex™, Virtex™-E
- Virtex™-II
- Virtex™-II Pro, Virtex™-II Pro X
- Virtex™-4
- Virtex™-5 LX
- Spartan™-II, Spartan™-IIE
- Spartan™-3, Spartan™-3E, Spartan™-3L
- CoolRunner™ XPLA3, CoolRunner™-II
- XC9500™, XC9500XL™, XC9500XV™

This chapter describes the IBISWriter program. This chapter contains the following sections:

IBISWriter Overview

The Input/Output Buffer Information Specification (IBIS) is a device modeling standard. IBIS allows for the development of behavioral models used to describe the signal behavior of device interconnects. These models preserve proprietary circuit information, unlike structural models such as those generated from SPICE (Simulation Program with Integrated Circuit Emphasis) simulations. IBIS buffer models are based on V/I curve data produced either by measurement or by circuit simulation.

IBIS models are constructed for each IOB standard, and an IBIS file is a collection of IBIS models for all I/O standards in the device. An IBIS file also contains a list of the used pins on a device that are bonded to IOBs configured to support a particular I/O standard (which associates the pins with a particular IBIS buffer model).

IBISWriter supports the use of digitally controlled impedance (DCI) for Virtex-II input designs with reference resistance that is selected by the user. Although it is not feasible to have IBIS models available for every possible user input, IBIS models are available for I/O Standards LVCMOS15 through LVCMOS33 for impedances of 40 and 65 ohms, in addition

to the 50 ohms impedance assumed by XCITE standards. If not specified, the default impedance value is 50 ohms.

The IBIS standard specifies the format of the output information file, which contains a file header section and a component description section. The *Golden Parser* has been developed by the IBIS Open Forum Group (http://www.eigroup.org/ibis) to validate the resulting IBIS model file by verifying that the syntax conforms to the IBIS data format.

The IBISWriter tool requires a design source file as input. For FPGA designs, this is a physical description of the design in the form of an NCD (native circuit description) file with a .ncd file extension. For CPLD designs, the input is produced by CPLDfit and has a .pnx file extension.

IBISWriter outputs a .ibs file. This file comprises a list of pins used by your design; the signals internal to the device that connect to those pins; and the IBIS buffer models for the IOBs connected to the pins.

Note: Virtex-II Pro architecture does not include the multi-gigabit transceiver (MGT). There are no IBIS models available for these IOBs.

To see an example of an IBIS file, refer to Virtex Tech Topic VTT004 at the following web location: http://www.xilinx.com/products/virtex/techtopic/vtt004.pdf

The following figure shows the IBISWriter flow:

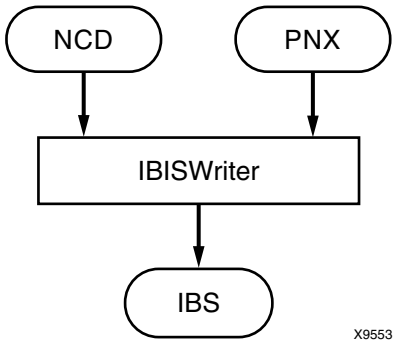


Figure 17-1: **IBISWriter Flow**

IBISWriter Syntax

Use the following syntax to run IBISWriter from the command line:

ibiswriter [*options*] *infile* *outfile*[**.ibs**]

options is one or more of the options listed in "IBISWriter Options".

infile is the design source file for the specified design. For FPGA designs, *infile* must have a .ncd extension. For CPLD designs, *infile* is produced by the CPLDfit and must have a .pnx extension.

outfile[.ibs] is the destination for the design specific IBIS file. The .ibs extension is optional. The length of the IBIS file name, including the .ibs extension, cannot exceed 24 characters.

IBISWriter Input Files

IBISWriter requires a design source file as input.

- FPGA Designs

  Requires a physical description of the design in the form of an NCD file with a .ncd file extension.

- CPLD Designs

  The input is produced by CPLDfit and has a .pnx file extension.

IBISWriter Output Files

IBISWriter outputs an .ibs ASCII file. This file comprises a list of pins used by your design, the signals internal to the device that connect to those pins, and the IBIS buffer models for the IOBs connected to the pins. The format of the IBIS output file is determined by the IBIS standard. The output file must be able to be validated by the Golden Parser to ensure that the file format conforms to the specification.

Note: IBISWriter gives an error message if a pin with an I/O Standard for which no buffer is available is encountered, or if a DCI value property is found for which no buffer model is available. After an error message appears, IBISWriter continues through the entire design, listing any other errors if and when they occur, then exiting without creating the .ibs output file. This error reporting helps users to identify problems and make corrections before running the program again.

IBISWriter Options

This section provides information on IBISWriter command line options.

–allmodels (Include all available buffer models for this architecture)

To reduce the size of the output .ibs file, IBISWriter produces an output file that contains only design-specific buffer models, as determined from the active pin list. For users who wish to access all available buffer models, the –allmodels command line option should be used.

Use the following syntax when using the –allmodels option:

ibiswriter –allmodels *infile outfile*.**ibs**

–g (Set Reference Voltage)

The –g command line option varies by architecture as shown in Table 17-1.

Use the following syntax when using the –g option:

ibiswriter –g *option_value_pair infile outfile*.**ibs**

The following is an example using the VCCIO:LVTTL option value pair.

ibiswriter –g VCCIO:LVTTL *design*.**ncd** *design*.**ibs**

The –g option supports only the architectures listed in the following table:

Table 17-1: **–g Options**

Architecture	Option	Value	Description
Virtex-E	OperatingConditions	Typical_Slow_Fast, Mixed	Use this option to set operating condition parameters. Typical_Slow_Fast refers to operating range defined by temperature, VCCIO, and manufacturing process ranges. If no –g option is given, the default value Typical_Slow_Fast is used.
Spartan-IIE	OperatingConditions	Typical_Slow_Fast,Mixed	Typical_ Slow_Fast refers to operating range defined by temperature, VCCIO, and manufacturing process ranges. If no –g option is given, the default value Typical_Slow_Fast is used.
XC9500	VCCIO	LVTTL, TTL	Use this option to configure I/Os for 3.3V (LVTTL) or 5V (TTL) VCCIO reference voltage. The –g option is required.
XC9500XL	VCCIO	LVCMOS2, LVTTL	Use this option to configure outputs for 3.3V (LVTTL) or 2.5V (LVCMOS2) VCCIO reference voltage. Each user pin is compatible with 5V, 3.3V, and 2.5V inputs. The –g option is required.

–intstyle (Integration Style)

-intstyle {**ise** | **xflow** | **silent**}

The –intstyle option reduces screen output based on the integration style you are running. When using the –intstyle option, one of three modes must be specified: *ise*, *xflow*, or *silent*. The mode sets the way information is displayed in the following ways:

-intstyle ise

This mode indicates the program is being run as part of an integrated design environment.

-intstyle xflow

This mode indicates the program is being run as part of an integrated batch flow.

-intstyle silent

This mode limits screen output to warning and error messages only.

Note: The -intstyle option is automatically invoked when running in an integrated environment, such as Project Navigator or XFLOW.

–ml (Multilingual Support)

The –ml option invokes the multilingual support feature to reference an external file (for example, a SPICE file).

–pin (Generate Package Parasitics)

The –pin option when enabled includes the per pin parasitics information, if available, for the given device-package combination. Some device-package combinations, like Virtex™-4, do not have this information available. When –pin is enabled for Virtex™-4 designs, IBISWriter adds a section to the output file that contains RLC parasitics (in a matrix format) for each package pin.

The –pin option is useful for analyzing timing and signal integrity. By default, this option is disabled.

Chapter 18

CPLDfit

CPLDfit is compatible with the following families:

- CoolRunner™ XPLA3, CoolRunner™-II
- XC9500™, XC9500XL™, XC9500XV™

This chapter describes the CPLDfit program. This chapter includes the following sections:

CPLDfit Overview

The CPLDfit program is a command line executable that takes a Native Generic Database (NGD) file, produced by NGDBuild, as input and fits the design into a CPLD device.

The following figure shows the CPLDfit design flow:

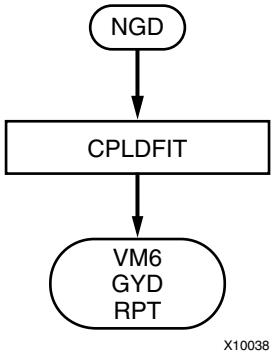


Figure 18-1: **CPLD Design Flow**

CPLDfit Syntax

Following is the command line syntax for running the CPLDfit program:

```
cpldfit infile.ngd [options]
```

options can be any number of the CPLDfit options listed in the "CPLDfit Options" section of this chapter. They do not need to be listed in any particular order. Separate multiple options with spaces.

CPLDfit Input Files

CPLDfit takes the following file as input:

- NGD file—Native Generic Database file output by NGDBuild. This file contains a logical description of the design expressed both in terms of the hierarchy used when the design was first created and in terms of lower-level Xilinx primitives to which the hierarchy resolves.

CPLDfit Output Files

CPLDfit outputs the following files:

- VM6 file—This file is the default output file from CPLDfit and the input file to the Hprep6 and TAEngine programs. See Chapter 21, "Hprep6" and Chapter 20, "TAEngine" for more information.
- GYD file—This file is the optional guide file generated by CPLDfit, which contains pin freeze information as well as the placement of internal equations from the last successful fit.
- RPT file—This file is the CPLDfit report file, which contains a resource summary, implemented equations, device pinout as well as the compiler options used by CPLDfit.
- XML file—This file is used to generate an HTML report.
- PNX file—This file is used by the IBISWriter program to generate an IBIS model for the implemented design.
- CXT file—This file is used by the XPower program to calculate and display power consumption. It is not available for XC9500/XL/XV devices.
- MFD file—This file is used by the ChipViewer GUI program and HTML Reports to generate a graphical representation of the design implementation.

CPLDfit Options

CPLDfit uses the following option files:

–blkfanin (Specify Maximum Fanin for Function Blocks)

```
-blkfanin [limit:4,40]
```

The -blkfanin option specifies the maximum number of function block inputs to use when fitting a device. If the value is near the maximum, this option reduces the possibility that design revisions will be able to fit without changing the pin-out. The maximum values vary with each supported CPLD architecture as shown below (default in parentheses):

CoolRunner XPLA3 = 40 (38)
CoolRunner-II = 40 (36)

–exhaust (Enable Exhaustive Fitting)

The values for inputs and pterms have an impact on design fitting. Occasionally different values must be tried before a design is optimally fit. The -exhaust option automates this process by iterating through all combinations of input and pterm limits until a fit is found. This process can take several hours depending on the size of the design. This option is off by default.

–ignoredatagate (Ignore DATA_GATE Attributes)

This option directs CPLDfit to ignore the DATA_GATE attribute when fitting a CoolRunner-II device. This option is off by default.

Architecture Support: CoolRunner-II

–ignoretspec (Ignore Timing Specifications)

CPLDfit optimizes paths to meet timing constraints. The -ignoretspec option directs CPLDfit to *not* perform this prioritized optimization. This option is off by default.

–init (Set Power Up Value)

```
-init [low|high|fpga]
```

The -init option specifies the default power up state of all registers. This option is overridden if an INIT attribute is explicitly placed on a register. Low and high are self-explanatory. The FPGA setting causes all registers with an asynchronous reset to power up low, all registers with an asynchronous preset to power up high, and remaining registers to power up low. The default setting is low.

–inputs (Number of Inputs to Use During Optimization)

```
-inputs [limit:2,36]
```

The -inputs option specifies the maximum number of inputs for a single equation. The higher this value, the more resources a single equation may use, possibly limiting the number of equations allowed in a single function block. The maximum limit varies with each CPLD architecture. The limits are as follows (default in parentheses):

XC9500 = 36 (36)
XC9500XL/XV = 54 (54)
CoolRunner XPLA3 = 40 (36)
CoolRunner-II = 40 (36)

–iostd (Specify I/O Standard)

```
-iostd [LVTTL|LVCMOS18|LVCMOS25 |SSTL2_I|SSTL3_I|HSTL_I|LVCMOS15]
```

The -iostd option sets the default voltage standard for all I/Os. The default is overridden by explicit assignments. Not all I/O standards are available for each architecture. The available I/O standards follow (default in parentheses):

CoolRunner-II: LVTTL, LVCMOS18, LVCMOS25, SSTL2_I, SSTL3_I, HSTL_I, LVCMOS15, LVCMOS18

–keepio (Prevent Optimization of Unused Inputs)

The -keepio option prevents unused inputs from being optimized. By default, CPLDfit trims unconnected input pins.

Architecture Support: XC9500, XC9500XL/XV, CoolRunner XPLA3, CoolRunner-II

–loc (Keep Specified Location Constraints)

```
-loc [on|off|try]
```

The -loc option specifies how CPLDfit uses the design location constraints. The *on* setting directs CPLDfit to obey location constraints. The *off* setting directs CPLDfit to ignore location constraints. The *try* setting directs CPLDfit to use location constraints unless doing so would result in a fitting failure. The default setting is *on*.

–localfbk (Use Local Feedback)

The XC9500 macrocell contains a local feedback path. The -localfbk option turns this feedback path on. This option is off by default.

Architecture Support: XC9500

–log (Specify Log File)

```
-log logfile
```

The -log option generates a logfile that contains all error, warning, and informational messages.

–nofbnand (Disable Use of Foldback NANDS)

This option disables the use of the Foldback Nand when fitting the design. This option is off by default.

Architecture Support: CoolRunner XPLA3

–nogclkopt (Disable Global Clock Optimization)

The -nogclkopt option turns off automatic global clock inferring. This option is off by default.

–nogsropt (Disable Global Set/Reset Optimization)

This option turns off automatic global set/reset inferring.

Architecture Support: XC9500, XC9500XL/XV, CoolRunner XPLA3, CoolRunner-II

–nogtsopt (Disable Global Output-Enable Optimization)

This option turns off automatic global 3-state inferring.

Architecture Support: XC9500, XC9500XL/XV, CoolRunner XPLA3, CoolRunner-II

–noisp (Turn Off Reserving ISP Pin)

The -noisp option disables the JTAG pins, allowing them to be used as I/O pins. This option is off by default.

Architecture Support: CoolRunner XPLA3

–nom1opt (Disable Multi-level Logic Optimization)

This -nomlopt option disables multi-level logic optimization when fitting a design. This option is off by default.

–nouim (Disable FASTConnect/UIM Optimization)

The XC9500 interconnect matrix allows multiple signals to be joined together to form a wired AND functionality. The -nouim option turns this functionality off. This option is off by default.

Architecture Support: XC9500

–ofmt (Specify Output Format)

```
-ofmt [vhdl|verilog|abel]
```

The -ofmt option sets the language used in the fitter report when describing implemented equations. The default is ABEL.

Architecture Support: XC9500, XC9500XL/XV, CoolRunner XPLA3, CoolRunner-II

–optimize (Optimize Logic for Density or Speed)

```
-optimize density|speed
```

This -optimize option directs CPLDfit to optimize the design for density or speed. Optimizing for density results in a slower speed but uses resource sharing to allow more logic to fit into a device. Optimizing for speed uses less resource sharing but flattens the logic, which results in fewer levels of logic (faster). Density is the default argument for this option.

–p (Specify Xilinx Part)

```
-p part
```

The -p option specifies the Xilinx product family; <part> is in the form of device-speedgrade-package (for example, XC2C512-10-FT256).

If only a product family is entered (for example, XPLA3), CPLDfit iterates through all densities until a fit is found.

–pinfbk (Use Pin Feedback)

The XC9500 architecture allows feedback into the device through the I/O pin. The -pinfbk option turns this feedback functionality on. This option is on by default.

Architecture Support: XC9500

–power (Set Power Mode)

```
-power [std|low|auto]
```

The -power option sets the default power mode of macrocells. This option can be overridden if a macrocell is explicitly assigned a power setting. The *std* setting is for standard high speed mode. The *low* setting is for low power mode (at the expense of speed). The *auto* setting allows CPLDfit to choose the *std* or *low* setting based on the timing constraints. The default setting for this option is *std.*

Note: This option is available for XC9500/XL/XV devices.

–pterms (Number of Pterms to Use During Optimization)

```
-pterms [limit:1,90]
```

The -pterms option specifies the maximum number of product terms for a single equation. The higher this value, the more product term resources a single equation may use, possibly limiting the number of equations allowed in a single function block. The maximum limit varies with each CPLD architecture. The limits are as follows (default in parenthesis):

XC9500 = 90 (25)
XC9500XL/XV = 90 (25)
CoolRunner XPLA3 = 48 (36)
CoolRunner-II = 56 (36)

–slew (Set Slew Rate)

```
-slew [fast|slow|auto]
```

The -slew option specifies the default slew rate for output pins. Fast and slow are self-explanatory. The auto setting allows CPLDfit to choose which slew rate to use based on the timing constraints. The default setting is fast.

–terminate (Set to Termination Mode)

```
-terminate [pullup|keeper|float]
```

The -terminate option globally sets all inputs and tristatable outputs to the specified form of termination. Not all termination modes exist for each architecture. The available modes for each architecture follow (default in parentheses):

XC9500 XL/ XV: Float, Keeper (keeper)
CoolRunner XPLA3: Float, Pullup (pullup)
CoolRunner-II: Float, Pullup, Keeper (float)

–unused (Set Termination Mode of Unused I/Os)

```
-unused [ground|pulldown|pullup|keeper|float]
```

The -unused option specifies how unused pins are terminated. Not all options are available for all architectures. The allowable options follow (default in parentheses):

XC9500 /XL/XV: Float, Ground (float)
CoolRunner XPLA3: Float, Pullup (pullup)
CoolRunner-II: Float, Ground, Pullup, Keeper (ground)

–wysiwyg (Do Not Perform Optimization)

The -wysiwyg option directs CPLDfit to *not* perform any optimization on the design provided to it. This option is off by default.

Chapter 19

TSIM

TSIM is compatible with the following families:

- CoolRunner™ XPLA3, CoolRunner™-II
- XC9500™, XC9500XL™, XC9500XV™

This chapter describes the TSIM program. This chapter includes the following sections:

TSIM Overview

The TSIM program is a command line executable that takes an implemented CPLD design file (VM6) as input and outputs an annotated NGA file used by the NetGen program. The NetGen Timing Simulation flow produces a back-annotated timing netlist for timing simulation. See the "CPLD Timing Simulation" section in Chapter 22, "NetGen" for more information.

TSIM Syntax

Following is the syntax for the TSIM command line program:

```
tsim design.vm6 output.nga
```

design.vm6 is the name of the input design file (VM6) output by the CPLDfit program. See Chapter 18, "CPLDfit" for more information.

output.nga is the name of the output file for use with the NetGen Timing Simulation flow to create a back-annotated netlist for timing simulation. If an output file name is not specified, TSIM uses the root name of the input design file with a .nga extension.

TSIM Input Files

TSIM uses the following file as input:

VM6 file—This file is a database file, output by CPLDfit, that contains the mapping of the user design into the target CPLD architecture.

TSIM Output Files

TSIM outputs the following file:

NGA file—This back-annotated logical design file is used as an input file for the NetGen Timing Simulation flow.

Chapter 20

TAEngine

TAEngine is compatible with the following families:

- CoolRunner™ XPLA3, CoolRunner™-II
- XC9500™, XC9500XL™, XC9500XV™

This chapter describes the Timing Analysis Engine (TAEngine) program. TAEngine is a command line executable that performs static timing analysis on implemented Xilinx CPLD designs. This chapter includes the following sections:

TAEngine Overview

TAEngine takes an implemented CPLD design file (VM6) from CPLDfit and performs a static timing analysis of the design's timing components. The results of the static timing analysis are written to a TAEngine report file (TIM) in summary or detail format.

The default output for TAEngine is a TIM report in summary format, which lists all timing paths and their delays. A detail formatted TIM report, specified with the "–detail (Detail Report)" option, lists all timing paths as well as a summary of all individual timing components that comprise each path. Both the summary and detail formatted TIM reports show the performance of all timing constraints contained in the design.

The following figure shows the design flow for the TAEngine program:

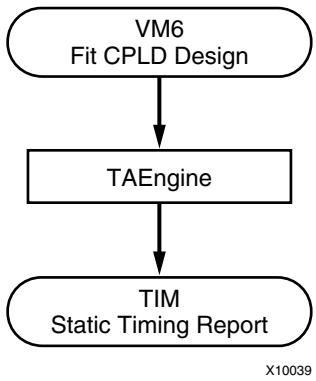


Figure 20-1: **TAEngine Design Flow**

TAEngine Syntax

Following is the command line syntax for running TAEngine:

```
taengine design_name.vm6 [options]
```

design_name is the name of the VM6 design file.

options can be any number of the TAEngine options listed in the "TAEngine Options" section of this chapter. They do not need to be listed in any particular order. Separate multiple options with spaces.

TAEngine Input Files

TAEngine takes the following file as input:

- VM6—An implemented CPLD design produced by the CPLDfit program.

TAEngine Output Files

TAEngine outputs the following file:

- TIM file—An ASCII (text) timing report file with a .tim extension that lists the timing paths and performance to timing constraints contained in the design. This report file can be produced in summary (default) or detail format.

TAEngine Options

This section describes the TAEngine command line options.

–detail (Detail Report)

–detail

The –detail option is used to produce a detail formatted TAEngine report (TIM) that shows static timing analysis for all paths in the design, as well as details for the delays in each path.

–iopath (Trace Paths)

–iopath

The –iopath option instructs TAEngine to trace paths through bi-directional pins.

–l (Specify Output Filename)

–l *output_file*.tim

The –l option specifies the name of the output report file. By default, the TAEngine takes the root name of the input design file and adds a .tim extension (*design_name*.tim).

Chapter 21

Hprep6

Hprep6 is compatible with the following families:

- CoolRunner™ XPLA3, CoolRunner™-II
- XC9500™, XC9500XL™, XC9500XV™

This chapter describes the Hprep6 program. Hprep6 is a command line executable that takes an implemented CPLD design file (VM6) as input and generates a programming file for configuring a Xilinx CPLD device. This chapter includes the following sections:

Hprep6 Overview

Hprep6 takes an implemented CPLD design file (VM6) from the CPLDfit program and generates a programming file for downloading to a CPLD device. Program files are generated in JEDEC (JED) format and optionally ISC format based on options specified on the command line.

The following figure shows the Hprep6 design flow.

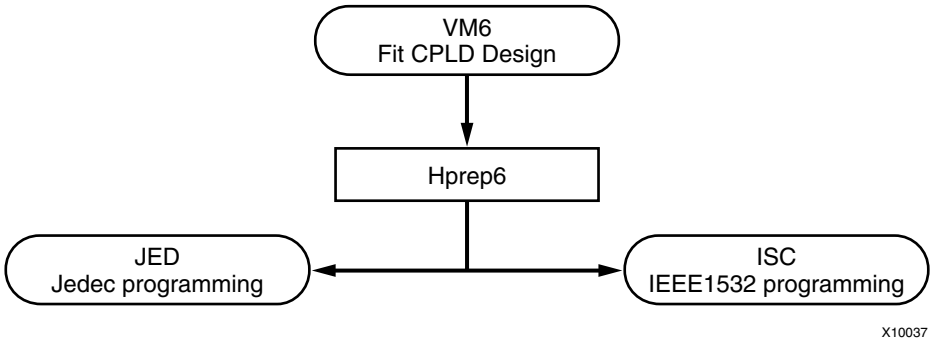


Figure 21-1: **Hprep6 Design Flow**

Hprep6 Syntax

Following is the command line syntax for running the Hprep6 program:

```
hprep6 -i design_name.vm6 [options]
```

design_name is the name of the input design file. The -i option is required to specify the input .vm6 file.

options can be any number of the Hprep6 options listed in the "Hprep6 Options" section of this chapter. They do not need to be listed in any particular order. Separate multiple options with spaces.

Hprep6 Input Files

Hprep6 uses the following file as input:

- VM6—An implemented CPLD design file from the CPLDfit command line program. See Chapter 18, "CPLDfit" for additional information.

Hprep6 Output Files

Hprep6 outputs the following files:

- JED file—A JEDEC file used for CPLD programming
- ISC file—A IEEE1532 file used for CPLD programming

Hprep6 Options

This section describes the Hprep6 command line options.

–autosig (Automatically Generate Signature)

The -autosig option allows the iMPACT configuration software to automatically generate a signature based on the checksum of the JEDEC file. If the -n <signature> option is supplied, -autosig is ignored.

Note: This option is applicable only to the XC9500/XL/XV families.

–intstyle (Integration Style)

```
-intstyle {ise | xflow | silent}
```

The –intstyle option reduces screen output based on the integration style you are running. When using the –intstyle option, one of three modes must be specified: *ise*, *xflow*, or *silent*. The mode sets the way information is displayed in the following ways:

-intstyle ise

This mode indicates the program is being run as part of an integrated design environment.

-intstyle xflow

This mode indicates the program is being run as part of an integrated batch flow.

-intstyle silent

This mode limits screen output to warning and error messages only.

–n (Specify Signature Value for Readback)

```
-n [signature]
```

The –signature option is applicable to the XC9500/XL/XV devices only. The value entered in the signature field programs a set of bits in the CPLD that may be read-back via JTAG after programming. This is often used as to identify the version of a design programmed into a device.

Note: The CoolRunner family also allows for a signature value, but it must be entered by the programming tool (for instance, IMPACT or third party programmer).

–nopullup (Disable Pullups)

The -nopullup option instructs Hprep6 to disable the pullups on empty function blocks. By default, pullups are enabled to minimize leakage current and prevent floating I/Os.

Note: The –nopullup option applies to XC9500/XL/XV devices only.

–s (Produce ISC File)

The –s option instructs Hprep6 to output an additional programming file in ISC IEEE1532 format.

Note: ISC IEEE532 output is not available for the CoolRunner XPLA3 family.

–tmv (Specify Test Vector File)

```
-tmv filename
```

The -tmv option is used to specify a test vector file for use with iMPACT's functional test operation. The output .tmv file is in ABEL format.

Chapter 22

NetGen

The NetGen program is compatible with the following families:

- Virtex™, Virtex™-E
- Virtex™-II
- Virtex™-II Pro, Virtex™-II Pro X
- Virtex™-4
- Virtex™-5 LX
- Spartan™-II, Spartan™-IIE
- Spartan™-3, Spartan™-3E, Spartan™-3L
- CoolRunner™ XPLA3, CoolRunner™-II
- XC9500™, XC9500XL™, XC9500XV™

This chapter describes the NetGen program and contains the following sections:

NetGen Overview

The NetGen application is a command line executable that reads Xilinx design files as input, extracts data from the design files, and generates netlists that are used with supported third-party simulation, equivalence checking, and static timing analysis tools.

NetGen supports the following flow types:

- Functional Simulation for FPGA and CPLD designs
- Timing Simulation for FPGA and CPLD designs
- Equivalence Checking for FPGA designs
- Static Timing Analysis for FPGA designs

The flow type that NetGen runs is based on the input design file (NGC, NGD, or NCD). The following table shows the output file types, based on the input design files:

Table 22-1: **NetGen Output File Types**

Input Design File	Output File Type
NGC	UNISIM-based functional simulation netlist
NGD	SIMPRIM-based functional netlist
NCA from CPLD	SIMPRIM-based netlist, along with a full timing SDF file.
NCD from MAP	SIMPRIM-based netlist, along with a partial timing SDF file
NCD from PAR	SIMPRIM-based netlist, along with a full timing SDF file

NetGen can take an implemented design file and write out a single netlist for the entire design, or multiple netlists for each module of a hierarchical design. Individual modules of a design can be simulated on their own, or together at the top-level. Modules identified with the KEEP_HIERARCHY attribute are written as user-specified Verilog, VHDL, and SDF netlists with the "–mhf (Multiple Hierarchical Files)"option. See "Preserving and Writing Hierarchy Files" for additional information.

The following figure outlines the NetGen flow for implemented FPGA designs.

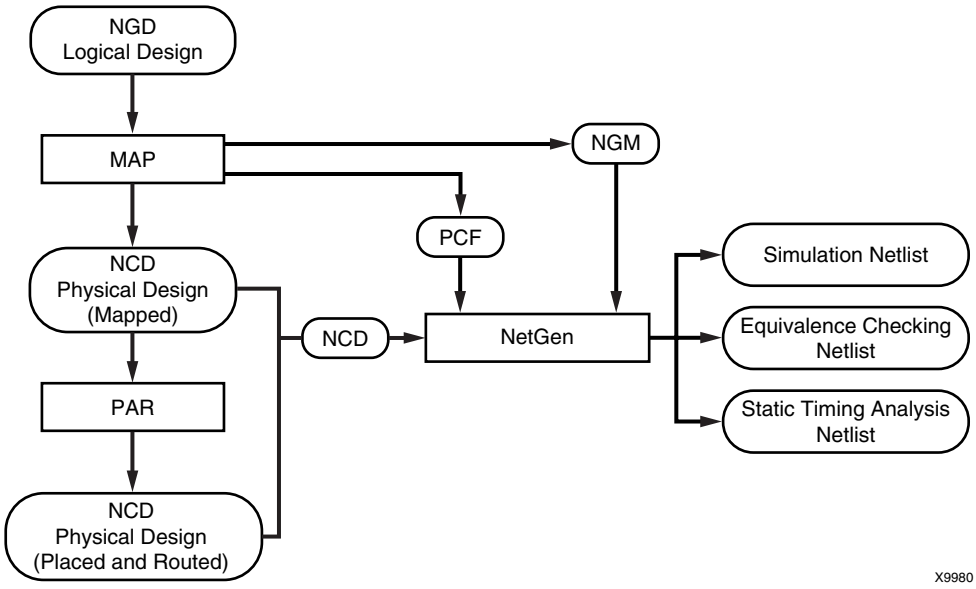


Figure 22-1: **NetGen Flow**

NetGen Supported Flows

NetGen can be described as having three fundamental flows: simulation, equivalency checking, and third-party static timing analysis. This chapter contains flow-specific sections that detail the use and features of NetGen support flows and describe any sub-flows. For example, the simulation flow includes two flows types: functional simulation and timing simulation.

Each flow-specific section includes command line syntax, input files, output files, and available command line options for each NetGen flow.

NetGen syntax is based on the type of NetGen flow you are running. For details on NetGen flows and syntax, refer to the flow-specific sections that follow.

Valid netlist flows are:

simulation [**sim**]— generates a simulation netlist for functional simulation or timing simulation. For this netlist type, you must specify the output file type as Verilog or VHDL with the –ofmt option.

```
netgen -sim [options]
```

equivalence [**ecn**]— generates a Verilog-based equivalence checking netlist. For this netlist type, you must specify a tool name after the **–**ecn option. Possible tool names for this netlist type are **conformal** or **formality**.

```
netgen -ecn conformal|formality [options]
```

static timing analysis [**sta**]—generates a Verilog netlist for static timing analysis.

```
netgen -sta [options]
```

NetGen Simulation Flow

Within the NetGen Simulation flow, there are two sub-flows: functional simulation and timing simulation. The functional simulation flow may be used for UNISIM-based or SIMPRIM-based netlists, based on the input file. An input NGC file will generate a UNISIM-based netlist for functional simulation. An input NGD file will generate a SIMPRIM-based netlist for functional simulation. Similarly, timing simulation can be broken down further to post-map timing simulation and post-par timing simulation, both of which use SIMPRIM-based netlists.

Note: NetGen does not list LOC parameters when an NGD file is used as input. In this case, “UNPLACED” is reported as the default value for LOC parameters.

Options for the NetGen Simulation flow (and sub-flows) can be viewed by running **netgen -h sim** from the command line.

NetGen Functional Simulation Flow

This section describes the functional simulation flow, which is used to translate NGC and NGD files into Verilog or VHDL netlists.

When you enter an NGC file as input on the NetGen command line, NetGen invokes the functional simulation flow to produce a UNISIM-based netlist. Similarly, when you enter an NGD file as input on the NetGen command line, NetGen invokes the functional simulation flow to produce a SIMPRIM-based netlist. You must also specify the type of netlist you want to create: Verilog or VHDL.

The Functional Simulation flow uses the following files as input:

- NGC —This file output by XST is used to create a UNISIM-based netlist suitable for using with IP Cores and performing post-synthesis functional simulation.
- NGD—This file output by NGDBuild contains a logical description of the design and is used to create a SIMPRIM-based netlist.

Notes on Functional Simulation for UNISIM-based Netlists

For XST users, the output NGC file can be entered on the command line. For third-party synthesis tool users, you must first use the ngcbuild command to convert all of the design netlists to a single NGC file, which NetGen takes as input.

The following command reads the top-level EDIF netlist and converts it to an NGC file:

ngcbuild [*options*] *top_level_netlist_file output_ngc_file*

Note: For information on NGCBuild, see Answer Record #21851 at http://www.xilinx.com/support.

Syntax for NetGen Functional Simulation

The following command runs the NetGen Functional Simulation flow:

```
netgen -ofmt {verilog|vhdl} [options] input_file[.ngd|ngc|ngo]
```

verilog or vhdl is the output netlist format that you specify with the required –ofmt option.

options is one or more of the options listed in the "Options for NetGen Simulation Flow" section. In addition to common options, this section also contains Verilog and VHDL-specific options.

input_file is the input file name and extension.

Output files for NetGen Functional Simulation

- V file—This is a IEEE 1364-2001 compliant Verilog HDL file that contains the netlist information obtained from the input design files. This file is a functional simulation model and cannot be synthesized or used in any manner other than simulation.
- VHD file—This VHDL IEEE 1076.4 VITAL-2000 compliant VHDL file contains the netlist information obtained from the input design files. This file is a simulation model and cannot be synthesized or used in any other manner than simulation.

NetGen Timing Simulation Flow

This section describes the NetGen Timing Simulation flow, which is used for timing verification on FPGA and CPLD designs. For FPGA designs, timing simulation is done after PAR, but may also be done after MAP if only component delay and no route delay information is needed. When performing timing simulation, you must specify the type of netlist you want to create: Verilog or VHDL. In addition to the specified netlist, NetGen also creates an SDF file as output. The output Verilog and VHDL netlists contain the functionality of the design and the SDF file contains the timing information for the design.

Input file types depend on whether you are using an FPGA or CPLD design. Please refer to "FPGA Timing Simulation" and "CPLD Timing Simulation" for design-specific information, including input file types.

A complete list of command line options for performing NetGen Timing Simulation appears at the end of this section.

Syntax for NetGen Timing Simulation

The following command runs the NetGen Timing Simulation flow:

```
netgen -sim -ofmt {verilog|vhdl} [options] input_file[.ncd]
```

verilog or vhdl is the output netlist format that you specify with the required –ofmt option.

options is one or more of the options listed in the "Options for NetGen Simulation Flow" section. In addition to common options, this section also contains Verilog and VHDL-specific options.

input_file is the input NCD file name and extension.

To get help on command line usage for NetGen Timing Simulation, type:

```
netgen -h sim
```

FPGA Timing Simulation

You can verify the timing of an FPGA design using the NetGen Timing Simulation flow to generate a Verilog or VHDL netlist and an SDF file. The figure below illustrates the NetGen Timing Simulation flow using an FPGA design.

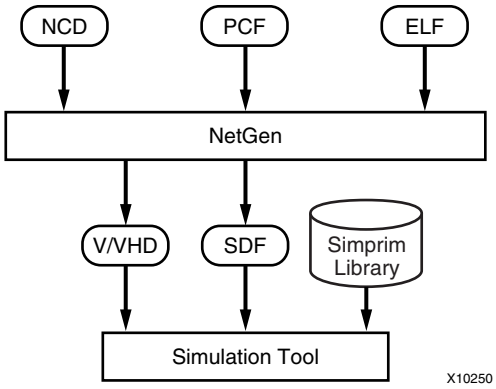


Figure 22-2: **FPGA Timing Simulation**

The FPGA Timing Simulation flow uses the following files as input:

- NCD —This physical design file may be mapped only, partially or fully placed, or partially or fully routed.
- PCF (optional)—This is a physical constraints file. If prorated voltage or temperature is applied to the design, the PCF must be included to pass this information to NetGen. See "–pcf (PCF File)" for more information.
- ELF (MEM) (optional)—This file populates the Block RAMs specified in the .bmm file. See "–bd (Block RAM Data File)" for more information.

Output files for FPGA Timing Simulation

- SDF file—This SDF 3.0 compliant standard delay format file contains delays obtained from the input design files.
- V file—This is a IEEE 1364-2001 compliant Verilog HDL file that contains the netlist information obtained from the input design files. This file is a simulation model of the implemented design and cannot be synthesized or used in any manner other than simulation.
- VHD file—This VHDL IEEE 1076.4 VITAL-2000 compliant VHDL file contains the netlist information obtained from the input design files. This file is a simulation model of the implemented design and cannot be synthesized or used in any other manner than simulation.

CPLD Timing Simulation

You can use the NetGen Timing Simulation flow to verify the timing of a CPLD design after it is implemented using CPLDfit and the delays are annotated using the –tsim option. The input file is the annotated NGA file from the TSIM program.

Note: See Chapter 18, "CPLDfit" and Chapter 19, "TSIM" for additional information.

The figure below illustrates the NetGen Timing Simulation flow using a CPLD design.

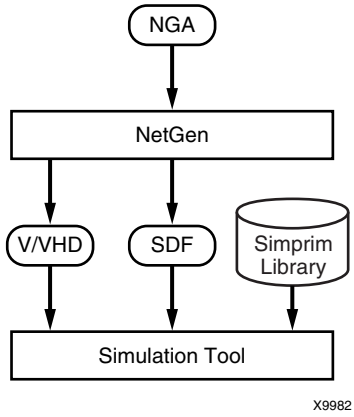


Figure 22-3: **CPLD Timing Simulation**

Input files for CPLD Timing Simulation

The CPLD Timing Simulation flow uses the following files as input:

- NGA file—This native generic annotated file is a logical design file from TSIM that contains Xilinx primitives. See Chapter 19, "TSIM" for additional information.

Output files for CPLD Timing Simulation

The NetGen Simulation Flow uses the following files as output:

- SDF file—This standard delay format file contains delays obtained from the input NGA file.

- V file—This is a IEEE 1364-2001 compliant Verilog HDL file that contains netlist information obtained from the input NGA file. This file is a simulation model of the fitted design and cannot be synthesized or used in any manner other than simulation.
- VHD file—This VHDL IEEE 1076.4 VITAL-2000 compliant VHDL file contains netlist information obtained from the input NGA file. This file is a simulation model of the fitted design and cannot be synthesized or used in any other manner than simulation.

Options for NetGen Simulation Flow

This section describes the supported NetGen command line options for timing simulation.

–aka (Write Also-Known-As Names as Comments)

-aka

The –aka option includes original user-defined identifiers as comments in the VHDL netlist. This option is useful if user-defined identifiers are changed because of name legalization processes in NetGen.

–bd (Block RAM Data File)

–bd *[filename]* [**.elf**|**.mem**] *[tag [tagname]}*

The –bd option specifies the path and file name of the .elf file used to populate the Block RAM instances specified in the .bmm file. The address and data information contained in the .elf (from EDK) or .mem file allows Data2MEM to determine which ADDRESS_BLOCK to place the data. Multiple use of the -bd option is allowed.

Optionally, a tagname can be specified with the –bd option. If a tagname is specified, only the address spaces with the same name in the .bmm file are used for translation, and all other data outside of the tagname address spaces are ignored. See Chapter 24, "Data2MEM" for additional information.

–dir (Directory Name)

-dir *[directory_name]*

The –dir option specifies the directory in which the output files are written.

–fn (Control Flattening a Netlist)

-fn

The –fn option outputs a flattened netlist. A flat netlist is without any design hierarchy.

–gp (Bring Out Global Reset Net as Port)

–gp *port_name*

The –gp option causes NetGen to bring out the global reset signal (which is connected to all flip-flops and latches in the physical design) as a port on the top-level design module. Specifying the port name allows you to match the port name you used in the frontend.

This option is used only if the global reset net is not driven. For example, if you include a STARTUP_VIRTEX component in a Virtex-E design, you should not enter the –gp option, because the STARTUP_VIRTEX component drives the global reset net.

Note: Do not use GR, GSR, PRLD, PRELOAD, or RESET as port names, because these are reserved names in the Xilinx software. This option is ignored by UNISIM-based flows, which use an NGC file as input.

–insert_pp_buffers (Insert Path Pulse Buffers)

–insert_pp_buffers true | false

The –insert_pp_buffers option controls whether path pulse buffers are inserted into the output netlist to eliminate pulse swallowing. Pulse swallowing is seen on signals in back-annotated timing simulations when the pulse width is shorter than the delay on the input port of the component. For example, if a clock of period 5 ns (2.5 ns high/2.5 ns low) is propagated through a buffer, but in the SDF, the PORT or IOPATH delay for the input port of that buffer is greater than 2.5 ns, the output will be unchanged in the waveform window (e.g., if the output was "X" at the start of simulation, it will remain at "X").

By default this command is set to false.

Note: This option is available when the input is an NCD file.

–intstyle (Integration Style)

-intstyle {**ise** | **xflow** | **silent**}

The –intstyle option reduces screen output based on the integration style you are running. When using the –intstyle option, one of three modes must be specified: *ise*, *xflow*, or *silent*. The mode sets the way information is displayed in the following ways:

-intstyle ise

This mode indicates the program is being run as part of an integrated design environment.

-intstyle xflow

This mode indicates the program is being run as part of an integrated batch flow.

-intstyle silent

This mode limits screen output to warning and error messages only.

Note: The -intstyle option is automatically invoked when running in an integrated environment, such as Project Navigator or XFLOW.

–mhf (Multiple Hierarchical Files)

The –mhf option is used to write multiple hierarchical files--one for every module that has the KEEP_HIERARCHY attribute.

Note: See "Preserving and Writing Hierarchy Files" for additional information.

–module (Simulation of Active Module)

-module

The –module option creates a netlist file based on the active module only, independent of the top-level design. NetGen constructs the netlist based only on the active module's interface signals.

To use this option you must specify an NCD file that contains an expanded active module.

Note: The –module option is for use with the Modular Design flow.

–ofmt (Output Format)

-ofmt verilog|vhdl

The –ofmt option is a required option that specifies output format of either Verilog or VHDL netlists.

–pcf (PCF File)

-pcf *pcf_file*.**pcf**

The –pcf option allows you to specify a PCF (physical constraints file) as input to NetGen. You only need to specify a physical constraints file if prorating constraints (temperature and/or voltage) are used.

Temperature and voltage constraints and prorated delays are described in the *Constraints Guide*.

Note: The –pcf option is valid for the timing simulation flow.

–s (Change Speed)

-s [*speed grade*]

The –s option instructs NetGen to annotate the device speed grade you specify to the netlist. The device *speed* can be entered with or without the leading dash. For example, both –s 3 and –s –3 are allowable entries.

Some architectures support the –s min option. This option instructs NetGen to annotate a process minimum delay, rather than a maximum worst-case to the netlist. The command line syntax is the following.

-s min

Minimum delay values may not be available for all families. Use the Speedprint or PARTGen utility programs in the software to determine whether process minimum delays are available for your target architecture. See Chapter 4, "PARTGen" and Chapter 13, "Speedprint" for additional information.

Settings made with the –s min option override any prorated timing parameters in the PCF. If –s min is used then all fields (MIN:TYP:MAX) in the resulting SDF file are set to the process minimum value.

Note: The –s option is valid for the timing simulation flow.

–sim (Generate Simulation Netlist)

The -sim option writes a simulation netlist. This is the default option for NetGen, and the default option for NetGen for generating a simulation netlist.

–tb (Generate Testbench Template File)

The –tb option generates a testbench file with a .tb extension. It is a ready-to-use Verilog or VHDL template file, based on the input NCD file. The type of template file (Verilog or VHDL) is specified with the –ofmt option.

–ti (Top Instance Name)

–ti *top_instance_name*

The –ti option specifies a user instance name for the design under test in the testbench file created with the –tb option.

–tm (Top Module Name)

–tm *top_module_name*

By default (without the –tm option), the output files inherit the top module name from the input NCD file. The –tm option changes the name of the top-level module name appearing in the NetGen output files.

–tp (Bring Out Global 3-State Net as Port)

–tp *port_name*

The –tp option causes NetGen to bring out the global 3-state signal (which forces all FPGA outputs to the high-impedance state) as a port on the top-level design module or output file. Specifying the port name allows you to match the port name you used in the front-end.

This option is only used if the global 3-state net is not driven. For example, if you include a STARTUP_VIRTEX component in an Virtex-E design, you should not have to enter a –tp option, because the STARTUP_VIRTEX component drives the global 3-state net.

Note: Do not use the name of any wire or port that already exists in the design, because this causes NetGen to issue an error. This option is ignored in UNISIM-based flows, which use an NGC file as input.

–w (Overwrite Existing Files)

The –w option causes NetGen to overwrite the .vhd or .v file if it exists. By default, NetGen does *not* overwrite the netlist file.

Note: All other output files are automatically overwritten.

Verilog-Specific Options for Functional and Timing Simulation

This section describes the Verilog-specific command line options for timing simulation.

–insert_glbl (Insert glbl.v Module)

–insert_glbl true | false

The –insert_glbl option specifies that the glbl.v module is included in the output Verilog simulation netlist. The default value of this option is true. The –insert_glbl option when set to false, specifies that the output Verilog netlist will not contain the glbl.v module. For more information on glbl.v, see the *Synthesis and Simulation Design Guide.*

Note: If the –mhf (multiple hierarchical files) option is used, –insert_glbl cannot be set to true.

–ism (Include SimPrim Modules in Verilog File)

The –ism option includes SimPrim modules from the SimPrim library in the output Verilog (.v) file. This option allows you to bypass specifying the library path during simulation. However, using this switch increases the size of your netlist file and increases your compile time.

When you run this option, NetGen checks that your library path is set up properly. Following is an example of the appropriate path:

```
$XILINX/verilog/src/simprim
```

If you are using compiled libraries, this switch offers no advantage. If you use this switch, do not use the –ul switch.

Note: The –ism option is valid for post-translate (NGD), post-map, and post-place and route simulation flows.

–ne (No Name Escaping)

By default (without the –ne option), NetGen "escapes" illegal block or net names in your design by placing a leading backslash (\) before the name and appending a space at the end of the name. For example, the net name "p1$40/empty" becomes "\p1$40/empty" when name escaping is used. Illegal Verilog characters are reserved Verilog names, such as "input" and "output," and any characters that do not conform to Verilog naming standards.

The –ne option replaces invalid characters with underscores so that name escaping does not occur. For example, the net name "p1$40/empty" becomes "p1$40_empty" when name escaping is not used. The leading backslash does not appear as part of the identifier. The resulting Verilog file can be used if a vendor's Verilog software cannot interpret escaped identifiers correctly.

–pf (Generate PIN File)

The –pf option writes out a pin file—a Cadence signal-to-pin mapping file with a .pin extension.

Note: NetGen only generates a PIN file if the input is an NGM file.

–sdf_anno (Include $sdf_annotate)

-sdf_anno [**true**|**false**]

The –sdf_anno option controls the inclusion of the $sdf_annotate construct in a Verilog netlist. The default for this option is true. To disable this option, use false.

Note: The –sdf_anno option is valid for the timing simulation flow.

–sdf_path (Full Path to SDF File)

-sdf_path [*path_name*]

The –sdf_path option outputs the SDF file to the specified full path. This option writes the full path and the SDF file name to the $sdf_annotate statement. If a full path is not specified, it writes the full path of the current work directory and the SDF file name to the $sdf_annotate statement.

Note: The –sdf_path option is valid for the timing simulation flow.

–shm (Write $shm Statements in Test Fixture File)

The -shm option places $shm statements in the structural Verilog file created by NetGen. These $shm statements allow NC-Verilog to display simulation data as waveforms. This option is for use with Cadence NC-Verilog files only.

–ul (Write uselib Directive)

The –ul option causes NetGen to write a library path pointing to the SimPrim library into the output Verilog (.v) file. The path is written as shown below:

```
uselib dir=$XILINX/verilog/src/simprims libext=.v
```

$XILINX is the location of the Xilinx software.

If you do not enter a –ul option, the ‘uselib line is not written into the Verilog file.

Note: A blank ‘uselib statement is automatically appended to the end of the Verilog file to clear out the ‘uselib data. If you use this option, do not use the –ism option.

Note: The –ul option is valid for SIMPRIM-based functional simulation and timing simulation flows; although not all simulators support the ‘uselib directive. Xilinx recommends using this option with caution.

–vcd (Write $dump Statements In Test Fixture File)

The –vcd option writes $dumpfile/$dumpvars statements in testfixture. This option is for use with Cadence Verilog files only.

VHDL-Specific Options for Functional and Timing Simulation

This section describes the VHDL-specific command line options for timing simulation.

–a (Architecture Only)

By default, NetGen generates both entities and architectures for the input design. If the –a option is specified, no entities are generated and only architectures appear in the output.

–ar (Rename Architecture Name)

`-ar` *architecture_name*

The –ar option allows you to rename the architecture name generated by NetGen. The default architecture name for each entity in the netlist is STRUCTURE.

–rpw (Specify the Pulse Width for ROC)

–rpw *roc_pulse_width*

The –rpw option specifies the pulse width, in nanoseconds, for the ROC component. You must specify a positive integer to simulate the component. This option is not required. By default, the ROC pulse width is set to 100 ns.

–tpw (Specify the Pulse Width for TOC)

–tpw *toc_pulse_width*

The –tpw option specifies the pulse width, in nanoseconds, for the TOC component. You must specify a positive integer to simulate the component. This option is required when you instantiate the TOC component (for example, when the global set/reset and global 3-State nets are sourceless in the design).

–xon (Select Output Behavior for Timing Violations)

–xon {**true**|**false**}

The –xon option specifies the output behavior when timing violations occur on memory elements. If you set this option to true, any memory elements that violate a setup time trigger X on the outputs. If you set this option to false, the signal's previous value is retained. If you do not set this option, –xon true is run.

Note: The –xon option should be avoided as much as possible. If there is an asynchronous path in the design, the constraint ASYNC_REG should be used. Disabling the X propagation globally can have detrimental results on the simulation and the timing simulation results may not match the behavior seen in the hardware. Please see the Disabling X propagation section in the *Synthesis and Simulation Design Guide* for more information.

NetGen Equivalence Checking Flow

This section describes the NetGen Equivalence Checking flow, which is used for formal verification of FPGA designs. This flow creates a Verilog netlist and conformal or formality assertion file for use with supported equivalence checking tools.

The figures below illustrate the NetGen Equivalence Checking flow for FPGA designs.

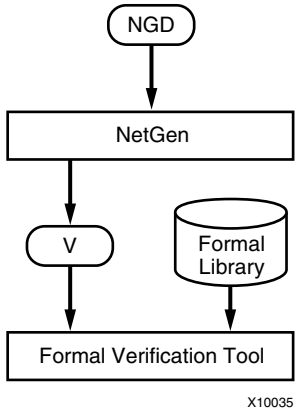


Figure 22-4: **Post-NGDBuild Flow for FPGAs**

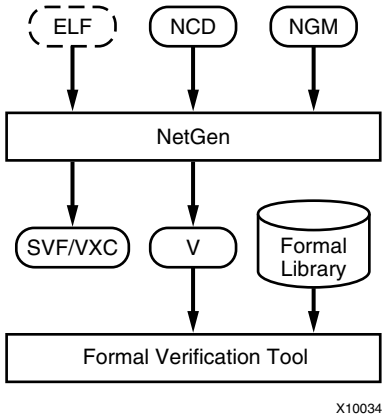


Figure 22-5: **Post-Implementation Flow for FPGAs**

Syntax for NetGen Equivalence Checking.

The following command runs the NetGen Equivalence Checking flow:

```
netgen -ecn [tool_name] [options] input_file[.ncd/.ngd] [ngm_file.ngm]
```

options is one or more of the options listed in the "Options for NetGen Equivalence Checking Flow" section.

tool_name is a required switch that generates a netlist compatible with equivalence checking tools. Valid *tool_name* arguments are **`conformal`** or **`formality`**. For additional information on equivalence checking and formal verification tools, please refer to the *Synthesis and Simulation Design Guide*.

input_file is the input NCD or NGD file. If an NGD file is used, the .ngd extension must be specified.

ngm_file (optional, but recommended) is the input NGM file, which is a design file, produced by MAP, that contains information about what was trimmed and transformed during the MAP process.

To get help on command line usage for NetGen Timing Simulation, type:

```
netgen -h ecn
```

Input files for NetGen Equivalence Checking

The NetGen Equivalence Checking flow uses the following files as input:

- NGD file—This file is a logical description of an unmapped FPGA design.
- NCD file—This physical design file may be mapped only, partially or fully placed, or partially or fully routed.
- NGM file —This mapped design file is generated by MAP and contains information on what was trimmed and transformed during the MAP process. See "–ngm (Design Correlation File)" for more information.
- ELF (MEM) (optional)—This file is used to populate the Block RAMs specified in the .bmm file. See "–bd (Block RAM Data File)" for more information.

Output files for NetGen Equivalence Checking

The NetGen Equivalence Checking flow uses the following files as output:

- Verilog (.v) file—This is a IEEE 1364-2001 compliant Verilog HDL file that contains the netlist information obtained from the input file. This file is a equivalence checking model and cannot be synthesized or used in any other manner than equivalence checking.
- Formality (.svf) file—This is an assertion file written for the Formality equivalence checking tool. This file provides information about some of the transformations that a design went through, after it was processed by Xilinx implementation tools.
- Conformal-LEC (.vxc) file—This is an assertion file written for the Conformal-LEC equivalence checking tool. This file provides information about some of the transformations that a design went through, after it was processed by Xilinx implementation tools.

Note: For specific information on Conformal-LEC and Formality tools, please refer to the *Synthesis and Simulation Design Guide*.

Options for NetGen Equivalence Checking Flow

This section describes the supported NetGen command line options for equivalence checking.

–aka (Write Also-Known-As Names as Comments)

The –aka option includes original user-defined identifiers as comments in the VHDL netlist. This option is useful if user-defined identifiers are changed because of name legalization processes in NetGen.

–bd (Block RAM Data File)

–bd *[filename]* [**.elf**|**.mem**] *[tag [tagname]}*

The –bd option specifies the path and file name of the .elf file used to populate the Block RAM instances specified in the .bmm file. The address and data information contained in the .elf (from EDK) or .mem file allows Data2MEM to determine which `ADDRESS_BLOCK` to place the data. Multiple use of the -bd option is allowed.

Optionally, a tagname can be specified with the –bd option. If a tagname is specified, only the address spaces with the same name in the .bmm file are used for translation, and all other data outside of the tagname address spaces are ignored. See Chapter 24, "Data2MEM" for additional information.

–dir (Directory Name)

-dir *[directory_name]*

The –dir option specifies the directory in which the output files are written.

–ecn (Equivalence Checking)

-ecn *[tool_name]* [**conformal**|**formality**]

The –ecn option generates an equivalence checking netlist. This option generates a file that can be used for formal verification of an FPGA design.

For additional information on equivalence checking and formal verification tools, please refer to the *Synthesis and Simulation Design Guide*.

–fn (Control Flattening a Netlist)

The –fn option produces a flattened netlist.

–intstyle (Integration Style)

-intstyle {**ise** | **xflow** | **silent**}

The –intstyle option reduces screen output based on the integration style you are running. When using the –intstyle option, one of three modes must be specified: *ise*, *xflow*, or *silent*. The mode sets the way information is displayed in the following ways:

-intstyle ise

This mode indicates the program is being run as part of an integrated design environment.

-intstyle xflow

This mode indicates the program is being run as part of an integrated batch flow.

-intstyle silent

This mode limits screen output to warning and error messages only.

Note: The -intstyle option is automatically invoked when running in an integrated environment, such as Project Navigator or XFLOW.

–mhf (Multiple Hierarchical Files)

The –mhf option is used to write multiple hierarchical files, one for every module that has the KEEP_HIERARCHY attribute.

–module (Verification of Active Module)

-module

The –module option creates a netlist file based on the active module, independent of the top-level design. NetGen constructs the netlist based only on the active module's interface signals.

To use this option you must specify an NCD file that contains an expanded active module.

Note: This option is for use with the Modular Design flow.

–ne (No Name Escaping)

By default (without the –ne option), NetGen "escapes" illegal block or net names in your design by placing a leading backslash (\) before the name and appending a space at the end of the name. For example, the net name "p1$40/empty" becomes "\p1$40/empty" when name escaping is used. Illegal Verilog characters are reserved Verilog names, such as "input" and "output," and any characters that do not conform to Verilog naming standards.

The –ne option replaces invalid characters with underscores, so that name escaping does not occur. For example, the net name "p1$40/empty" becomes "p1$40_empty" when name escaping is not used. The leading backslash does not appear as part of the identifier. The resulting Verilog file can be used if a vendor's Verilog software cannot interpret escaped identifiers correctly.

–ngm (Design Correlation File)

-ngm [*ngm_file*]

The -ngm option is used to specify an NGM design correlation file. This option is used for equivalence checking flows.

–tm (Top Module Name)

–tm *top_module_name*

By default (without the –tm option), the output files inherit the top module name from the input NCD or NGM file. The –tm option changes the name of the top-level module name appearing within the NetGen output files.

–w (Overwrite Existing Files)

The –w option causes NetGen to overwrite the .v file if it exists. By default, NetGen does *not* overwrite the netlist file.

Note: All other output files are automatically overwritten.

NetGen Static Timing Analysis Flow

This section describes the NetGen Static Timing Analysis flow, which is used for analyzing the timing, including minimum of maximum delay values, of FPGA designs.

Minimum of maximum delays are used by static timing analysis tools to calculate skew, setup and hold values. Minimum of maximum delays are the minimum delay values of a device under a specified operating condition (speed grade, temperature and voltage). If the operating temperature and voltage are not specified, then the worst case temperature and voltage values are used. Note that the minimum of maximum delay value is different from the process minimum generated by using the **-s min** option.

The following example shows DELAY properties containing relative minimum and maximum delays.

Note: Both the TYP and MAX fields contain the maximum delay.

```
(DELAY)
(ABSOLUTE)
(PORT I (234:292:292) (234:292:292))
(IOPATH I O (392:489:489) (392:489:489))
```

Note: Timing simulation does not contain any relative delay information, instead the MIN, TYP, and MAX fields are all equal.

NetGen uses the Static Timing Analysis flow to generate Verilog and SDF netlists compatible with supported static timing analysis tools.

The figure below illustrates the NetGen Static Timing Analysis flow.

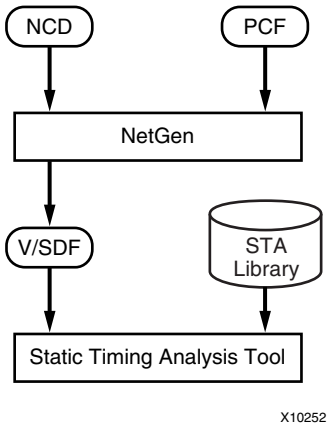


Figure 22-6: **Static Timing Analysis Flow for FPGAs**

Input files for Static Timing Analysis

The Static Timing Analysis flow uses the following files as input:

- NCD file—This physical design file may be mapped only, partially or fully placed, or partially or fully routed.
- PCF (optional)—This is a physical constraints file. If prorated voltage and temperature is applied to the design, the PCF file must be included to pass this information to NetGen. See "–pcf (PCF File)" for more information.

Output files for Static Timing Analysis

The Static Timing Analysis flow uses the following files as output:

- SDF file—This SDF 3.0 compliant standard delay format file contains delays obtained from the input file.
- Verilog (.v) file—This is a IEEE 1364-2001 compliant Verilog HDL file that contains the netlist information obtained from the input file. This file is a timing simulation model and cannot be synthesized or used in any manner other than for static timing analysis. This netlist uses simulation primitives, which may not represent the true implementation of the device. The netlist represents a functional model of the implemented design.

Syntax for NetGen Static Timing Analysis

The following command runs the NetGen Static Timing Analysis flow:

```
netgen -sta input_file[.ncd]
```

The *input_file* is the input file name and extension.

To get help on command line usage for equivalence checking, type:

```
netgen -h sta
```

Options for NetGen Static Timing Analysis Flow

This next section describes the supported NetGen command line options for static timing analysis.

–aka (Write Also-Known-As Names as Comments)

The –aka option includes original user-defined identifiers as comments in the Verilog netlist. This option is useful if user-defined identifiers are changed because of name legalization processes in NetGen.

–bd (Block RAM Data File)

–bd *[filename]* [**.elf**|**.mem**]

The –bd switch specifies the path and file name of the .elf file used to populate the Block RAM instances specified in the .bmm file. The address and data information contained in the .elf file allows Data2MEM to determine which ADDRESS_BLOCK to place the data.

–dir (Directory Name)

-dir *[directory_name]*

The -dir option specifies the directory in which the output files are written.

–fn (Control Flattening a Netlist)

The –fn option produces a flattened netlist.

–intstyle (Integration Style)

-intstyle {**ise** | **xflow** | **silent**}

The –intstyle option reduces screen output based on the integration style you are running. When using the –intstyle option, one of three modes must be specified: *ise*, *xflow*, or *silent*. The mode sets the way information is displayed in the following ways:

-intstyle ise

This mode indicates the program is being run as part of an integrated design environment.

-intstyle xflow

This mode indicates the program is being run as part of an integrated batch flow.

-intstyle silent

This mode limits screen output to warning and error messages only.

Note: The -intstyle option is automatically invoked when running in an integrated environment, such as Project Navigator or XFLOW.

–mhf (Multiple Hierarchical Files)

The –mhf option is used to write multiple hierarchical files, one for every module that has the KEEP_HIERARCHY attribute.

–module (Simulation of Active Module)

–module

The –module option creates a netlist file based on the active module, independent of the top-level design. NetGen constructs the netlist based only on the active module's interface signals.

To use this option you must specify an NCD file that contains an expanded active module.

Note: The –module option is for use with the Modular Design flow.

–ne (No Name Escaping)

By default (without the –ne option), NetGen "escapes" illegal block or net names in your design by placing a leading backslash (\) before the name and appending a space at the end of the name. For example, the net name "p1$40/empty" becomes "\p1$40/empty" when name escaping is used. Illegal Verilog characters are reserved Verilog names, such as "input" and "output," and any characters that do not conform to Verilog naming standards.

The –ne option replaces invalid characters with underscores, so that name escaping does not occur. For example, the net name "p1$40/empty" becomes "p1$40_empty" when name escaping is not used. The leading backslash does not appear as part of the identifier. The resulting Verilog file can be used if a vendor's Verilog software cannot interpret escaped identifiers correctly.

–pcf (PCF File)

-pcf *pcf_file*.**pcf**

The –pcf option allows you to specify a PCF (physical constraints file) as input to NetGen. You only need to specify a constraints file if it contains prorating constraints (temperature or voltage).

Temperature and voltage constraints and prorated delays are described in the *Constraints Guide*.

–s (Change Speed)

–s [*speed grade]*

The –s option instructs NetGen to annotate the device speed grade you specify to the netlist. The device *speed* can be entered with or without the leading dash. For example, –s 3 or –s –3 can be used.

Some architectures support the –s min option. This option instructs NetGen to annotate a process minimum delay, rather than a maximum worst-case delay and relative minimum delay, to the netlist. The command line syntax is the following:

-s min

Minimum delay values may not be available for all device families. Use the Speedprint program or the PARTGen program to determine whether process minimum delays are available for your target architecture. See Chapter 13, "Speedprint" or Chapter 4, "PARTGen" for more information.

Note: Settings made with the –s min option override any prorated timing parameters in the PCF. If –s min is used then all fields (MIN:TYP:MAX) in the resulting SDF file are set to the process minimum value.

–sta (Generate Static Timing Analysis Netlist)

The –sta option writes a static timing analysis netlist.

–tm (Top Module Name)

–tm *top_module_name*

By default (without the –tm option), the output files inherit the top module name from the input NCD file. The –tm option changes the name of the top-level module name appearing within the NetGen output files.

–w (Overwrite Existing Files)

The –w option causes NetGen to overwrite the .v file if it exists. By default, NetGen does *not* overwrite the netlist file.

All other output files are automatically overwritten.

Preserving and Writing Hierarchy Files

When hierarchy is preserved during synthesis and implementation using the KEEP_HIERARCHY constraint, the NetGen –mhf option writes separate netlists and SDF files (if applicable) for each piece of hierarchy.

The hierarchy of STARTUP and glbl (Verilog only) modules is preserved in the output netlist. If the -mhf option is used and there is at least one hierarchical block with the KEEP_HIERARCHY constraint in the design, NetGen writes out a separate netlist file for the STARTUP and glbl modules. If there is no block with the KEEP_HIERARCHY constraint, the -mhf option is ignored even if there are STARTUP and glbl modules in the design.

This section describes the output file types produced with the –mhf option. The type of netlist output by NetGen, depends on whether you are running the NetGen simulation, equivalence checking, or static timing analysis flow. For simulation, NetGen outputs a Verilog or VHDL file. The –ofmt option must be used to specify the output file type you wish to produce when you are running the NetGen simulation flow.

Note: When Verilog is specified, the $sdf_annotate is included in the Verilog netlist for each module.

The following table lists the base naming convention for hierarchy output files:

Table 22-2: **Hierarchy File Content**

Hierarchy File Content	Simulation	Equivalence Checking	Static Timing Analysis
File with Top-level Module	[*input_filename*] (default), or user specified output filename	[*input_filename*].ecn, or user specified output filename	[*input_filename*].sta, or user specified output filename
File with Lower Level Module	[*module_name*].sim	[*module_name*].ecn	[*module_name*].sta

The [*module_name*] is the name of the hierarchical module from the front-end that the user is already familiar with. There are cases when the [*module_name*] could differ, they are:

- If multiple instances of a module are used in the design, then each instantiation of the module is unique because the timing for the module is different. The names are made unique by appending an underscore followed by a "INST_" string and a count value (e.g., numgen, numgen_INST_1, numgen_INST_2).
- If a new filename clashes with an existing filename within the name scope, then the new name will be [*module_name*]_[*instance_name*].

Testbench File

A testbench file is created for the top-level design when the -tb option is used. The base name of the testbench file is the same as the base name of the design, with a .tv extension for Verilog, and a .tvhd extension for VHDL.

Hierarchy Information File

In addition to writing separate netlists, NetGen also generates a separate text file comprised of hierarchy information. The following information appears in the hierarchy text file. NONE appears if one of the files does not have relative information.

```
// Module      : The name of the hierarchical design module.
// Instance    : The instance name used in the parent module.
// Design File : The name of the file that contains the module.
// SDF File    : The SDF file associated with the module.
// SubModule   : The sub module(s) contained within a given module.
// Module, Instance : The sub module and instance names.
```

Note: The hierarchy information file for a top-level design does not contain an Instance field.

The base name of the hierarchy information file is:

[*design_base_name*]**_mhf_info.txt**

The STARTUP block is only supported on the top-level design module. The global set reset (GSR) and global tristate signal (GTS) connectivity of the design is maintained as described in the "Dedicated Global Signals in Back-Annotation Simulation" section of this chapter.

Dedicated Global Signals in Back-Annotation Simulation

The global set reset (GSR), PRLD for CPLDs, signal and global tristate signal (GTS) are global routing nets present in the design that provide a means of setting, resetting, or tristating applicable components in the device. The simulation behavior of these signals is modeled in the library cells of the Xilinx Simprim library and the simulation netlist using the glbl module in Verilog and the X_ROC / X_TOC components in VHDL.

The following sections explain the connectivity for Verilog and VHDL netlists.

Global Signals in Verilog Netlist

For Verilog, the glbl module is used to model the default behavior of global the GSR and GTS. The glbl.GSR and glbl.GTS can be directly referenced as global GSR/GST signals anywhere in a design or in any library cells.

NetGen writes out the glbl module definition in the output Verilog netlist. For a non-hierarchical design or a single-file hierarchical design, this glbl module definition is written at the bottom of the netlist. For a single-file hierarchical design, the glbl module is defined inside the top-most module. For a multi-file hierarchical design (-mhf option), NetGen writes out glbl.v as a hierarchical module.

If the GSR and GTS are brought out to the top-level design as ports using the -gp and -tp options, the top-most module has the following connectivity:

```
glbl.GSR = GSR_PORT
glbl.GTS = GTS_PORT
```

The GSR_PORT and GTS_PORT are ports on the top-level module created with the -gp and -tp options. If a STARTUP block is used in the design, the STARTUP block is translated to buffers that preserve the intended connectivity of the user-controlled signals to the global GSR and GTS (glbl.GSR and glbl.GTS).

When there is a STARTUP block in the design, the STARTUP block hierarchical level is always preserved in the output netlist. The output of STARTUP is connected to the global GSR/GTS signals (glbl.GSR and glbl.GTS).

For all hierarchical designs, the glbl module must be compiled and referenced along with the design. For information on setting the GSR and GTS for FPGAs, see the *"Simulating Verilog"* section in the *Synthesis and Simulation Design Guide*.

Global Signals in VHDL Netlist

Global signals for VHDL netlists are GSR and GTS, which are declared in the library package Simprim_Vcomponents.vhd. The GSR and GTS can be directly referenced anywhere in a design or in any library cells.

The X_ROC and X_TOC components in the VHDL library model the default behavior of the GSR and GTS. If the -gp and -tp options are not used, NetGen instantiates X_ROC and X_TOC in the output netlist. Each design has only one instance of X_ROC and X_TOC. For hierarchical designs, X_ROC and X_TOC are instantiated in the top-most module netlist.

X_ROC and X_TOC are instantiated as shown below:

```
X_ROC (O => GSR);
X_TOC (O => GTS);.
```

If the GSR and GTS are brought out to the top-level design using the -gp and -tp options, There will be no X_ROC or X_TOC instantiation in the design netlist. Instead, the top-most module has the following connectivity:

```
GSR<= GSR_PORT
GTS<= GTS_PORT
```

The GSR_PORT and GTS_PORT are ports on the top-level module created with the -gp and -tp options.

When there is a STARTUP block in the design, the STARTUP block hierarchical level is preserved in the output netlist. The output of STARTUP is connected to the global GSR and GTS signals.

For information on setting GSR and GTS for FPGAs, see the “Simulating VHDL” section of the *Synthesis and Simulation Design Guide*.

Chapter 23

XFLOW

XFLOW is compatible with the following families:

- Virtex™, Virtex™-E
- Virtex™-II
- Virtex™-II Pro, Virtex™-II Pro X
- Virtex™-4
- Virtex™-5 LX
- Spartan™-II, Spartan™-IIE
- Spartan™-3, Spartan™-3E, Spartan™-3L
- CoolRunner™ XPLA3, CoolRunner™-II
- XC9500™, XC9500XL™, XC9500XV™

This chapter describes the XFLOW program, a scripting tool that allows you to automate implementation, simulation, and synthesis flows using Xilinx programs. It contains the following sections:

XFLOW Overview

XFLOW is a command line program that automates Xilinx synthesis, implementation, and simulation flows. XFLOW reads a design file as input as well as a flow file and an option file. Xilinx provides a default set of flow files that automate which Xilinx programs are run to achieve a specific design flow. For example, a flow file can specify that NGDBuild, MAP, PAR, and TRACE are run to achieve an implementation flow for an FPGA. You can use the default set of flow files as is, or you can customize them. See "XFLOW Flow Types" and "Flow Files" for more information. Option files specify which command line options are run for each of the programs listed in the flow file. You can use the default set of option files provided by Xilinx, or you can create your own option files. See "XFLOW Options" for more information.

The following figure shows the inputs and the possible outputs of the XFLOW program. The output files depend on the flow you run.

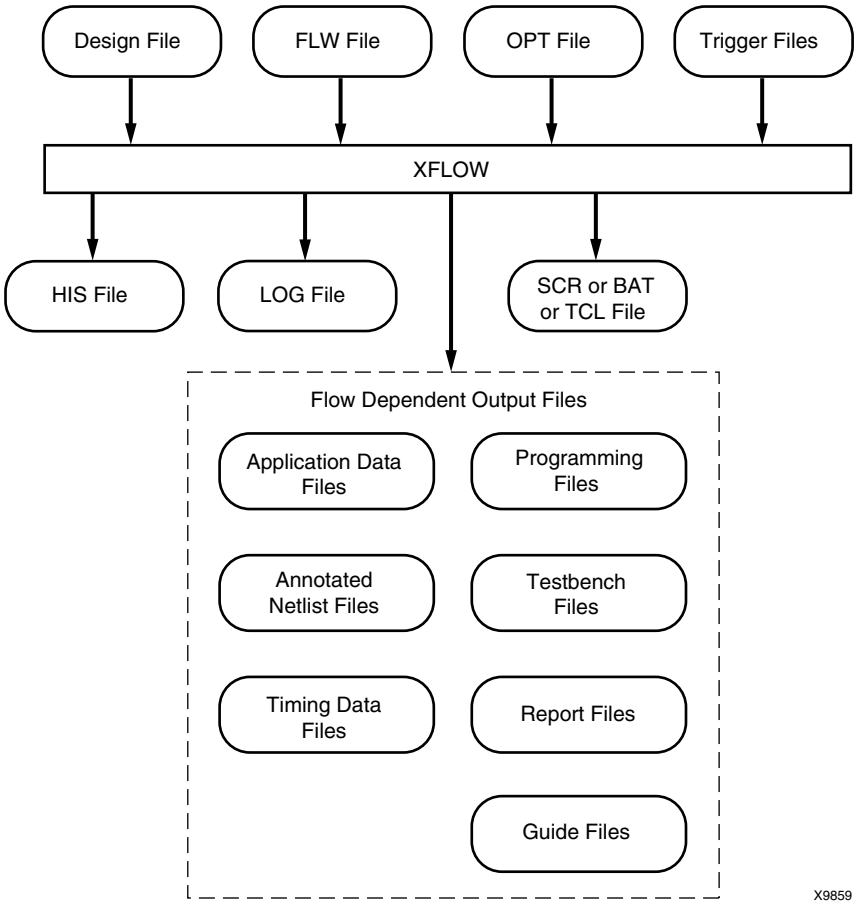


Figure 23-1: **XFLOW Design Flow**

XFLOW Syntax

Following is the command line syntax for XFLOW:

xflow `[-p partname]`[*flow type*]`[`*option file*`[`**.opt**`]]` `[`*xflow options*`]` *design_name*

flow type can be any of the flow types listed in "XFLOW Flow Types". Specifying a flow type prompts XFLOW to read a certain flow file. You can combine multiple flow types on one command line, but each flow type must have its own option file.

option file can be any of the option files that are valid for the specified flow type. See "XFLOW Option Files" for more information. In addition, option files are described in the applicable flow type section.

xflow options can be any of the options described in "XFLOW Options". They can be listed in any order. Separate multiple options with spaces.

design_name is the name of the top-level design file you want to process. See "XFLOW Input Files" for a description of input design file formats.

Note: If you specify a design name only and do not specify a flow type or option file, XFLOW defaults to the –implement flow type and fast_runtime.opt option file for FPGAs and the –fit flow type and balanced.opt option file for CPLDs.

You do not need to specify the complete path for option files. By default, XFLOW uses the option files in your working directory. If the option files are not in your working directory, XFLOW searches for them in the following locations and copies them to your working directory. If XFLOW cannot find the option files in any of these locations, it issues an error message.

- Directories specified using XIL_XFLOW_PATH
- Installed area specified with the XILINX environment variable

Note: By default, the directory from which you invoked XFLOW is your working directory. If you want to specify a different directory, use the –wd option described in "–wd (Specify a Working Directory)".

XFLOW Input Files

XFLOW uses the following files as input:

- Design File (for non-synthesis flows)—For all flow types except –synth, the input design can be an EDIF 2 0 0, or NGC (XST output) netlist file. You can also specify an NGD, NGO, or NCD file if you want to start at an intermediate point in the flow. XFLOW recognizes and processes files with the extensions shown in the following table.

File Type	Recognized Extensions
EDIF	.sedif, .edn, .edf, .edif
NCD	.ncd
NGC	.ngc
NGD	.ngd
NGO	.ngo

- Design File (for synthesis flows)—For the –synth flow type, the input design can be a Verilog or VHDL file. If you have multiple VHDL or Verilog files, you can use a PRJ or V file that references these files as input to XFLOW. For information on creating a PRJ or V file, see "Example 1: How to Synthesize VHDL Designs Using Command Line Mode" or "Example 2: How to Synthesize Verilog Designs Using Command Line Mode" of the *Xilinx Synthesis Technology (XST) User Guide*. You can also use existing PRJ files generated while using Project Navigator. XFLOW recognizes and processes files with the extensions shown in the following table.

File Type	Recognized Extensions
PRJ	.prj
Verilog	.v
VHDL	.vhd

Note: You must use the –g option for multiple file synthesis with Synplicity or Leonardo Spectrum. See "–synth" for details.

- FLW File—The flow file is an ASCII file that contains the information necessary for XFLOW to run an implementation or simulation flow. When you specify a flow type (described in "XFLOW Flow Types"), XFLOW calls a particular flow file. The flow file contains a program block for each program invoked in the flow. It also specifies the directories in which to copy the output files. You can use the default set of flow files as is, or you can modify them. See "Flow Files" for more information.
- OPT Files—Option files are ASCII files that contain options for each program included in a flow file. You can create your own option files or use the ones provided by Xilinx. See "XFLOW Option Files" for more information.
- Trigger Files—Trigger files are any additional files that a command line program reads as input, for example, UCF, NCF, PCF, and MFP files. Instead of specifying these files on the command line, these files must be listed in the Triggers line of the flow file. See "XFLOW Flow Types" for more information.

XFLOW Output Files

XFLOW always outputs the following files and writes them to your working directory.

- HIS file—The xflow.his file is an ASCII file that contains the XFLOW command you entered to execute the flow, the flow and option files used, the command line commands of programs that were run, and a list of input files for each program in the flow.
- LOG file—The xflow.log file is an ASCII file that contains all the messages generated during the execution of XFLOW.
- SCR, BAT, or TCL file—This script file contains the command line commands of all the programs run in a flow. This file is created for your convenience, in case you want to review all the commands run, or if you want to execute the script file at a later time. The file extension varies depending on your platform. The default outputs are SCR for UNIX and BAT for PC, although you can specify which script file to output by using the **`$scripts_to_generate`** variable.

In addition, XFLOW outputs one or more of the files shown in the following tables. The output files generated depend on the programs included in the flow files and the commands included in the option files.

Note: Report files are written to the working directory by default. You can specify a different directory by using the XFLOW –rd option, described in "–rd (Copy Report Files)", or by using the Report Directory option in the flow file, described in "Flow Files". All report files are in ASCII format.

The following table lists files that can be generated for both FPGA and CPLD designs.

Table 23-1: **XFLOW Output Files (FPGAs and CPLDs)**

File Name	Description	To Generate this File...
design_name.bld	This report file contains information about the NGDBuild run, in which the input netlist is translated to an NGD file.	Flow file must include "ngdbuild" (Use the –implement or –fit flow type)
time_sim.sdf *func_sim*.sdf	This Standard Delay Format file contains the timing data for a design.	Flow file must include "netgen" (Use the –tsim or –fsim flow type) Input must be an NGA file, which includes timing information
time_sim.tv *func_sim*.tv	This is an optional Verilog test fixture file.	Flow file must include "netgen" (Use the –tsim or –fsim flow type)
time_sim.tvhd *func_sim*.tvhd	This is an optional VHDL testbench file.	Flow file must include "netgen" (Use the –tsim or –fsim flow type)
time_sim.v *func_sim*.v	This Verilog netlist is a simulation netlist expressed in terms of Xilinx simulation primitives. It differs from the Verilog input netlist and should only be used for simulation, not implementation.	Flow file must include "netgen" (Use the –tsim or –fsim flow type)
time_sim.vhd *func_sim*.vhd	This VHDL netlist is a simulation netlist expressed in terms of Xilinx simulation primitives. It differs from the VHDL input netlist and should only be used for simulation, not implementation.	Flow file must include "netgen" (Use the –tsim or –fsim flow type)

The following table lists the output files that can be generated for FPGAs.

Table 23-2: **XFLOW Output Files (FPGAs)**

File Name	Description	To Generate this File...
design_name.bgn	This report file contains information about the BitGen run, in which a bitstream is generated for Xilinx device configuration.	Flow file must include "bitgen" (Use the –config flow type)
design_name.bit	This bitstream file contains configuration data that can be downloaded to an FPGA using PromGen, or iMPACT.	Flow file must include "bitgen" (Use the –config flow type)
design_name.dly	This report file lists delay information for each net in a design.	Flow file must include "par" (Use the –implement flow type)
design_name.ll	This optional ASCII file describes the position of latches, flip-flops, and IOB inputs and outputs in the BIT file.	Flow file must include "bitgen" (Use the –config flow type) Option file must include BitGen –l option
design_name.mrp	This report file contains information about the MAP run, in which a logical design is mapped to a Xilinx FPGA.	Flow file must include "map" (Use the –implement flow type)
design_name.ncd (by PAR phase) *design_name*_map.ncd (by MAP phase)	This Native Circuit Description file can be used as a guide file. It is a physical description of the design in terms of the components in the target Xilinx device. This file can be a mapped NCD file or a placed and routed NCD file.	Flow file must include "map" or "par" (Use the –implement flow type)
design_name.par	This report file contains summary information of all placement and routing iterations.	Flow file must include "par" (Use the –implement flow type)
design_name.pad	This report file lists all I/O components used in the design and their associated primary pins.	Flow file must include "par" (Use the –implement flow type)
design_name.rbt	This optional ASCII "rawbits" file contains ones and zeros representing the data in the bitstream file.	Flow file must include "bitgen" (Use the –config flow type) Option file must include BitGen –b option
design_name.twr	This report file contains timing data calculated from the NCD file.	Flow file must include "trce" (Use the –implement flow type)
design_name.xpi	This report file contains information on whether the design routed and timing specifications were met.	Flow file must include "par" (Use the –implement flow type)

The following table lists the output files that can be generated for CPLDs.

Table 23-3: **XFLOW Output Files (CPLDs)**

File Name	Description	To Generate this File...
design_name.gyd	This ASCII file is a CPLD guide file.	Flow file must include "cpldfit" (Use the –fit flow type)
design_name.jed	This ASCII file contains configuration data that can be downloaded to a CPLD using iMPACT.	Flow file must include "hprep6" (Use the –fit flow type)
design_name.rpt	This report file contains information about the CPLDfit run, in which a logical design is fit to a CPLD.	Flow file must include "cpldfit" (Use the –fit flow type)
design_name.tim	This report file contains timing data.	Flow file must include "taengine" (Use the –fit flow type)

XFLOW Flow Types

A "flow" is a sequence of programs invoked to synthesize, implement, simulate, and configure a design. For example, to implement an FPGA design the design is run through the NGDBuild, MAP, and PAR programs.

"Flow types" instruct XFLOW to execute a particular flow as specified in the relative flow file. (For more information on flow files, see, "Flow Files".) You can enter multiple flow types on the command line to achieve a desired flow. This section describes the flow types you can use.

Note: All flow types require that an option file be specified. If you do not specify an option file, XFLOW issues an error.

–assemble (Module Assembly)

```
-assemble option_file -pd pim_directory_path
```

Note: This flow type supports FPGA device families only.

This flow type runs the final phase of the Modular Design flow. In this "Final Assembly" phase, the team leader assembles the top-level design and modules into one NGD file and then implements the file.

Note: Use of this option assumes that you have completed the Initial Budgeting and Active Implementation phases of Modular Design. See "–implement (Implement an FPGA)" and "–initial (Initial Budgeting of Modular Design)" for details.

This flow type invokes the fpga.flw flow file and runs NGDBuild to create the NGD file that contains logic from the top-level design and each of the Physically Implemented Modules (PIMs). XFLOW then implements the NGD file by running MAP and PAR to create a fully expanded NCD file.

The working directory for this flow type should be the top-level design directory. You can either run the –assemble flow type from the top-level directory or use the –wd option to specify this directory. Specify the path to the PIMs directory after the –pd option. If you do not use the –pd option, XFLOW searches the working directory for the PIM files.The input design file should be the NGO file for the top-level design.

Xilinx provides the following option files for use with this flow type. These files allow you to optimize your design based on different parameters.

Table 23-4: **Option Files for –assemble Flow Type**

Option Files	Description
fast_runtime.opt	Optimized for fastest runtimes at the expense of design performance Recommended for medium to slow speed designs
balanced.opt	Optimized for a balance between speed and high effort
high_effort.opt	Optimized for high effort at the expense of longer runtimes Recommended for creating designs that operate at high speeds

The following example shows how to assemble a Modular Design with a top-level design named "top":

xflow –p xc2v250fg256-5 –assemble balanced.opt –pd ../pims top.ngo

–config (Create a BIT File for FPGAs)

–config *option_file*

This flow type creates a bitstream for FPGA device configuration using a routed design. It invokes the fpga.flw flow file and runs the BitGen program.

Xilinx provides the bitgen.opt option file for use with this flow type.

To use a netlist file as input, you must use the –implement flow type with the –config flow type. The following example shows how to use multiple flow types to implement and configure an FPGA:

xflow –p xc2v250fg256-5 –implement balanced.opt –config bitgen.opt testclk.edf

To use this flow type without the –implement flow type, you must use a placed and routed NCD file as input.

–ecn (Create a File for Equivalence Checking)

-ecn *option_file*

This flow type generates a file that can be used for formal verification of an FPGA design. It invokes the fpga.flw flow file and runs NGDBuild and NetGen to create a *netgen*.ecn file. This file contains a Verilog netlist description of your design for equivalence checking.

Xilinx provides the following option files for use with this flow type.

Table 23-5: **Option Files for –ecn Flow Type**

Option Files	Description
conformal_verilog.opt	Option file for equivalence checking for conformal
formality_verilog.opt	Option file for equivalence checking for formality

–fit (Fit a CPLD)

–fit *option_file*

This flow type incorporates logic from your design into physical macrocell locations in a CPLD. It invokes the cpld.flw flow file and runs NGDBuild and CPLDfit to create a JED file.

Xilinx provides the following option files for use with this flow type. These files allow you to optimize your design based on different parameters.

Table 23-6: **Option Files for –fit Flow Type**

Option Files	Description
balanced.opt	Optimized for a balance between speed and density
speed.opt	Optimized for speed
density.opt	Optimized for density

The following example shows how to use a combination of flow types to fit a design and generate a VHDL timing simulation netlist for a CPLD.

xflow -p xc2c64-4-cp56 -fit balanced.opt -tsim generic_vhdl.opt main_pcb.edn

–fsim (Create a File for Functional Simulation)

–fsim *option_file*

Note: The –fsim flow type can be used alone or with the –synth flow type. It cannot be combined with the –implement, –tsim, –fit, or –config flow types.

This flow type generates a file that can be used for functional simulation of an FPGA or CPLD design. It invokes the fsim.flw flow file and runs NGDBuild and NetGen to create a *func_sim*.edn, *func_sim*.v, or *func_sim*.vhdl file. This file contains a netlist description of your design in terms of Xilinx simulation primitives. You can use the functional simulation file to perform a back-end simulation with a simulator.

Xilinx provides the following option files, which are targeted to specific vendors, for use with this flow type.

Table 23-7: **Option Files for –fsim Flow Type**

Option File	Description
generic_vhdl.opt	Generic VHDL
modelsim_vhdl.opt	Modelsim VHDL
generic_verilog.opt	Generic Verilog
modelsim_verilog.opt	Modelsim Verilog
nc_verilog.opt	NC Verilog
verilog_xl.opt	Verilog-XL
vcs_verilog.opt	VCS Verilog
nc_vhdl.opt	NC VHDL
scirocco_vhdl.opt	Scirocco VHDL

The following example shows how to generate a Verilog functional simulation netlist for an FPGA design.

xflow -p xc2v250fg256-5 -fsim generic_verilog.opt `testclk.v`

–implement (Implement an FPGA)

–implement *option_file*

This flow type implements your design. It invokes the fpga.flw flow file and runs NGDBuild, MAP, PAR, and then TRACE. It outputs a placed and routed NCD file.

Xilinx provides the following option files for use with this flow type. These files allow you to optimize your design based on different parameters.

Table 23-8: **Option Files for –implement Flow Type**

Option Files	Description
fast_runtime.opt	Optimized for fastest runtimes at the expense of design performance Recommended for medium to slow speed designs
balanced.opt	Optimized for a balance between speed and high effort
high_effort.opt	Optimized for high effort at the expense of longer runtimes Recommended for creating designs that operate at high speeds
overnight.opt	Multi-pass place and route (MPPR) overnight mode

Table 23-8: **Option Files for –implement Flow Type**

Option Files	Description
weekend.opt	Multi-pass place and route (MPPR) weekend mode
exhaustive.opt	Multi-pass place and route (MPPR) exhaustive mode

The following example shows how to use the –implement flow type:

```
xflow -p xc2v250fg256-5 -implement balanced.opt testclk.edf
```

–initial (Initial Budgeting of Modular Design)

–initial budget.opt

Note: This flow type supports FPGA device families only.

This flow type runs the first phase of the Modular Design flow. In this "Initial Budgeting" phase, the team leader generates an NGO and NGD file for the top-level design. The team leader then sets up initial budgeting for the design. This includes assigning top-level timing constraints as well as location constraints for various resources, including each module.

This flow type invokes the fpga.flw flow file and runs NGDBuild to create an NGO and NGD file for the top-level design with all of the instantiated modules represented as unexpanded blocks. After running this flow type, assign constraints for your design using the Floorplanner and Constraints Editor tools.

Note: You cannot use the NGD file produced by this flow for mapping.

The working directory for this flow type should be the top-level design directory. You can either run the –initial flow type from the top-level design directory or use the –wd option to specify this directory. The input design file should be an EDIF netlist or an NGC netlist from XST. If you use an NGC file as your top-level design, be sure to specify the .ngc extension as part of your design name.

Xilinx provides the budget.opt option file for use with this flow type.

The following example shows how to run initial budgeting for a modular design with a top-level design named "top":

xflow –p xc2v250fg256-5 –initial budget.opt top.edf

–module (Active Module Implementation)

–module *option_file* **–active** *module_name*

Note: This flow type supports FPGA device families only. You *cannot* use NCD files from previous software releases with Modular Design in the current release. You must generate new NCD files with the current release of the software.

This flow type runs the second phase of the Modular Design flow. In this "Active Module Implementation" phase, each team member creates an NGD file for his or her module, implements the NGD file to create a Physically Implemented Module (PIM), and publishes the PIM using the PIMCreate command line tool.

This flow type invokes the fpga.flw flow file and runs NGDBuild to create an NGD file with just the specified "active" module expanded. This output NGD file is named after the top-level design. XFLOW then runs MAP and PAR to create a PIM.

Then, you must run PIMCreate to publish the PIM to the PIMs directory. PIMCreate copies the local, implemented module file, including the NGO, NGM and NCD files, to the appropriate module directory inside the PIMs directory and renames the files to *module_name.extension*. To run PIMCreate, type the following on the command line or add it to your flow file:

pimcreate *pim_directory* **-ncd** *design_name*_**routed.ncd**

The working directory for this flow type should be the active module directory. You can either run the –module flow type from the active module directory or use the –wd option to specify this directory. This directory should include the active module netlist file and the top-level UCF file generated during the Initial Budgeting phase. You must specify the name of the active module after the –active option, and use the top-level NGO file as the input design file.

Xilinx provides the following option files for use with this flow type. These files allow you to optimize your design based on different parameters.

Table 23-9: **Option Files for –module Flow Type**

Option Files	Description
fast_runtime.opt	Optimized for fastest runtimes at the expense of design performance Recommended for medium to slow speed designs
balanced.opt	Optimized for a balance between speed and high effort
high_effort.opt	Optimized for high effort at the expense of longer runtimes Recommended for designs that operate at high speeds

The following example shows how to implement a module.

xflow –p xc2v250fg256-5 –module balanced.opt –active controller ~teamleader/mod_des/implemented/top/top.ngo

–mppr (Multi-Pass Place and Route for FPGAs)

–mppr *option_file*

This flow type runs multiple place and route passes on your FPGA design. It invokes the fpga.flw flow file and runs NGDBuild, MAP, multiple PAR passes, and TRACE. After running the multiple PAR passes, XFLOW saves the "best" NCD file in the subdirectory called mppr.dir. (Do not change the name of this default directory.) This NCD file uses the naming convention *placer_level_router_level_cost_table*.ncd.

XFLOW then copies this "best" result to the working directory and renames it *design_name*.ncd. It also copies the relevant DLY, PAD, PAR, and XPI files to the working directory.

Note: By default, XFLOW does not support the multiple-node feature of the PAR Turns Engine. If you want to take advantage of this UNIX-specific feature, you can modify the appropriate option file to include the PAR –m option. See "–m (Multi-Tasking Mode)" in Chapter 9 for more information.

Xilinx provides the following option files for use with this flow type. These files allow you to set how exhaustively PAR attempts to place and route your design.

Note: Each place and route iteration uses a different "cost table" to create a different NCD file. There are 100 cost tables numbered 1 through 100. Each cost table assigns weighted values to relevant factors such as constraints, length of connection, and available routing resources.

Table 23-10: **Option Files for –mppr Flow Type**

Option Files	Description
overnight.opt	Runs 10 place and route iterations
weekend.opt	Runs place and route iterations until the design is fully routed or until 100 iterations are complete
exhaustive.opt	Runs 100 place and route iterations

The following example shows how to use the **-**mppr flow type:

```
xflow -p xc2v250fg256-5 -mppr overnight.opt testclk.edf
```

–sta (Create a File for Static Timing Analysis)

–sta *option_file*

This flow type generates a file that can be used to perform static timing analysis of an FPGA design. It invokes the fpga.flw flow file and runs NGDBuild and NetGen to generate a Verilog netlist compatible with supported static timing analysis tools.

Xilinx provides the following option file for use with this flow type.

Table 23-11: **Option Files for –sta Flow Type**

Option File	Description
primetime_verilog.opt	Option file for static timing analysis of Primetime.

–synth

–synth *option_file*

Note: When using the –synth flow type, you must specify the –p option.

This flow type allows you to synthesize your design for implementation in an FPGA, for fitting in a CPLD, or for compiling for functional simulation. The input design file can be a Verilog or VHDL file.

You can use the -synth flow type alone or combine it with the -implement, -fit, or -fsim flow type. If you use the -synth flow type alone, XFLOW invokes either the fpga.flw or cpld.flw file and runs XST to synthesize your design. If you combine the -synth flow type with the -implement, -fit, or -fsim flow type, XFLOW invokes the appropriate flow file, runs XST to synthesize your design, and processes your design as described in one of the following sections:

- "–implement (Implement an FPGA)"
- "–fit (Fit a CPLD)"
- "–fsim (Create a File for Functional Simulation)"

Synthesis Types

There are three different synthesis types that are described in the following sections.

XST

Use the following example to enter the XST command:

```
xflow -p xc2v250fg256-5 -synth xst_vhdl.opt design_name.vhd
```

If you have multiple VHDL or Verilog files, you can use a PRJ file that references these files as input. Use the following example to enter the PRJ file:

```
xflow -p xc2v250fg256-5 -synth xst_vhdl.opt design_name.prj
```

Leonardo Spectrum

Use the following example to enter the Leonardo Spectrum command:

```
xflow -p xc2v250fg256-5 -synth leonardospectrum_vhdl.opt
design_name.vhd
```

If you have multiple VHDL files, you must list all of the source files in a text file, one per line and pass that information to XFLOW using the –g (Specify a Global Variable) option. Assume that the file that lists all source files is `filelist.txt` and `design_name.vhd` is the top level design. Use the following example:

```
xflow -p xc2v250fg256-5 -g srclist:filelist.txt -synth
leonardospectrum_vhdl.opt design_name.vhd
```

The same rule applies for Verilog too.

Synplicity

Use the following example to enter the Synplicity command:

```
xflow -p xc2v250fg256-5 -synth synplicity_vhdl.opt design_name.vhd
```

If you have multiple VHDL files, you must list all the source files in a text file, one per line and pass that information to XFLOW using the –g (Specify a Global Variable) option. Assume that the file that lists all source files is `filelist.txt` and `design_name.vhd` is the top level design. Use the following example:

```
xflow -p xc2v250fg256-5 -g srclist:filelist.txt -synth
synplicity_vhdl.opt design_name.vhd
```

The same rule applies for Verilog too.

The following example shows how to use a combination of flow types to synthesize and implement a design:

```
xflow -p xc2v250fg256-5 -synth xst_vhdl.opt -implement balanced.opt
testclk.prj
```

Option Files for -synth Flow Types

Xilinx provides the following option files for use with the –synth flow type. These files allow you to optimize your design based on different parameters.

Table 23-12: **Option Files for –synth Flow Type**

Option File	Description
xst_vhdl.opt leonardospectrum_vhdl.opt synplicity_vhdl.opt	Optimizes a VHDL source file for speed, which reduces the number of logic levels and increases the speed of the design
xst_verilog.opt leonardospectrum_verilog.opt synplicity_verilog.opt	Optimizes a Verilog source file for speed, which reduces the number of logic levels and increases the speed of the design
xst_mixed.opt	Optimizes a mixed level VHDL and Verilog source file for speed, which reduces the number of logic levels and increases the speed of the design.

The following example shows how to use a combination of flow types to synthesize and implement a design:

```
xflow –p xc2v250fg256-5 –synth xst_vhdl.opt -implement balanced.opt
testclk.prj
```

–tsim (Create a File for Timing Simulation)

–tsim *option_file*

This flow type generates a file that can be used for timing simulation of an FPGA or CPLD design. It invokes the fpga.flw or cpld.flw flow file, depending on your target device. For FPGAs, it runs NetGen. For CPLDs, it runs TSim and NetGen. This creates a time_sim.v or time_sim.vhdl file that contains a netlist description of your design in terms of Xilinx simulation primitives. You can use the output timing simulation file to perform a back-end simulation with a simulator.

Xilinx provides the following option files, which are targeted to specific vendors, for use with this flow type.

Table 23-13: **Option Files for –tsim Flow Type**

Option File	Description
generic_vhdl.opt	Generic VHDL
modelsim_vhdl.opt	Modelsim VHDL
generic_verilog.opt	Generic Verilog
modelsim_verilog.opt	Modelsim Verilog
scirocco_vhdl.opt	Scirocco VHDL
nc_verilog.opt	NC Verilog
verilog_xl.opt	Verilog-XL
vcs_verilog.opt	VCS Verilog
nc_vhdl.opt	NC VHDL

The following example shows how to use a combination of flow types to fit and perform a VHDL timing simulation on a CPLD:

```
xflow -p xc2c64-4-cp56 -fit balanced.opt -tsim generic_vhdl.opt
main_pcb.vhd
```

Flow Files

When you specify a flow type on the command line, XFLOW invokes the appropriate flow file and executes some or all of the programs listed in the flow file. These files have a .flw extension. Programs are run in the order specified in the flow file.

Xilinx provides three flow files. You can edit these flow files, to add a new program, modify the default settings, and add your own commands between Xilinx programs. However, you cannot create new flow files of your own.

The following table lists the flow files invoked for each flow type.

Table 23-14: **Xilinx Flow Files**

Flow Type	Flow File	Devices	Flow Phase	Programs Run
–synth	fpga.flw	FPGA	Synthesis	XST, Synplicity, Leonardo Spectrum
–initial			Modular Design Initial Budgeting Phase	NGDBuild
–module			Modular Design Active Module Implementation Phase	NGDBuild, MAP, PAR
–assemble			Modular Design Final Assembly Phase	NGDBuild, MAP, PAR
–implement			Implementation	NGDBuild, MAP, PAR, TRACE
–mppr			Implementation (with Multi-Pass Place and Route)	NGDBuild, MAP, PAR (multiple passes), TRACE
–tsim			Timing Simulation	NGDBuild, NetGen
–ecn			Equivalence Checking	NGDBuild, NetGen
–sta			Static Timing Analysis	NGDBuild, NetGen
–config			Configuration	BitGen
–synth	cpld.flw	CPLD	Synthesis	XST, Synplicity, Leonardo Spectrum
–fit			Fit	NGDBuild, CPLDfit, TAEngine, HPREP6
–tsim			Timing Simulation	TSim, NetGen
–synth	fsim.flw	FPGA/ CPLD	Synthesis	XST, Synplicity, Leonardo Spectrum
–fsim			Functional Simulation	NGDBuild, NetGen

Flow File Format

The flow file is an ASCII file that contains the following information:

Note: You can use variables for the file names listed on the Input, Triggers, Export, and Report lines. For example, if you specify **Input: <design>.vhd** on the Input line, XFLOW automatically reads the VHDL file in your working directory as the input file.

- ExportDir

  This section specifies the directory in which to copy the output files of the programs in the flow. The default directory is your working directory.

  Note: You can also specify the export directory using the –ed command line option. The command line option overrides the ExportDir specified in the flow file.

- ReportDir

  This section specifies the directory in which to copy the report files generated by the programs in the flow. The default directory is your working directory.

  Note: You can also specify the report directory using the –rd command line option. The command line option overrides the ReportDir specified in the flow file.

- Global user-defined variables

  This section allows you to specify a value for a global variable, as shown in the following example:

```
Variables
$simulation_output = time_sim;
End variables
```

The flow file contains a program block for each program in the flow. Each program block includes the following information:

- **Program** *program_name*

  This line identifies the name of the program block. It also identifies the command line executable if you use an executable name as the *program_name*, for example, ngdbuild. This is the first line of the program block.

- **Flag: ENABLED | DISABLED**
  - ENABLED: This option instructs XFLOW to run the program if there are options in the options file.
  - DISABLED: This option instructs XFLOW to *not* run the program even if there are corresponding options in the options file.

- **Input:** *filename*

  This line lists the name of the input file for the program. For example, the NGDBuild program block might list design.edn.

- **Triggers:**

  This line lists any additional files that should be read by the program. For example, the NGDBuild program block might list design.ucf.

- **Exports:**

  This line lists the name of the file to export. For example, the NGDBuild program block might list design.ngd.

- **Reports:**

  This line lists the report files generated. For example, the NGDBuild program block might list design.bld.

- **Executable:** *executable_name*

  This line is optional. It allows you to create multiple program blocks for the same program. When creating multiple program blocks for the same program, you must enter a name other than the program name in the Program line (for example, enter post_map_trace, not trce). In the Executable line, you enter the name of the program as you would enter it on the command line (for example, trce).

For example, if you want to run TRACE after MAP and again after PAR, the program blocks for post-MAP TRACE and
post-PAR TRACE appear as follows:

```
Program post_map_trce
Flag: ENABLED;
Executable: trce;
Input: <design>_map.ncd;
Exports: <design>.twr, <design>.tsi;
End Program post_map_trce

Program post_par_trce
Flag: ENABLED;
Executable: trce;
Input: <design>.ncd;
Reports: <design>.twr, <design>.tsi;
End Program post_par_trce
```

Note: If your option file includes a corresponding program block, its Program line must match the Program line in the flow file (for example, post_map_trace).

- **End Program** *program_name*

  This line identifies the end of a program block. The *program_name* should be consistent with the *program_name* specified on the line that started the program block.

User Command Blocks

To run your own programs in the flow, you can add a "user command block" to the Flow File. The syntax for a user command block is the following:

```
UserCommand
     Cmdline: <user_cmdline>;
End UserCommand
```

Following is an example:

```
UserCommand
     Cmdline: "myscript.csh";
End UserCommand
```

Note: You cannot use the asterisk (*) dollar sign ($) and parentheses ( ) characters as part of your command line command.

XFLOW Option Files

Option files contain the options for all programs run in a flow. These files have an .opt extension. Xilinx provides option files for each flow type, as described in the different sections of "XFLOW Flow Types". You can also create your own option files.

Note: If you want to create your own option files, Xilinx recommends that you make a copy of an existing file, rename it, and then modify it.

Option File Format

Option files are in ASCII format. They contain program blocks that correspond to the programs listed in the flow files. Option file program blocks list the options to run for each program. Program options can be command line options or parameter files.

- Command Line Options

  For information on the different command line options for each program, see the program-specific chapters of this guide, or from the command line type the program name followed by –h on the command line. Some options require that you specify a particular file or value.

- Parameter files

  Parameter files specify parameters for a program. Parameters are written into the specified file. For example, Xilinx Synthesis Technology (XST) uses a script file to execute its command line options:

```
Program xst
       -ifn <design>_xst.scr;
       -ofn <design>_xst.log;
       ParamFile: <design>_xst.scr
           "run";
           "-ifn <synthdesign>";
           "-ifmt Verilog";
           "-ofn <design>.ngc";
.
.
.
       End ParamFile
End Program xst
```

Note: You can use variables for the file names listed in the Option Files. For example, if you specify **<design>.vhd** as an input file, XFLOW automatically reads the VHDL file in your working directory as the input file.

XFLOW Options

This section describes the XFLOW command line options. These options can be used with any of the flow types described in the preceding section.

–active (Active Module)

–active *active_module*

The –active option specifies the active module for Modular Design; "active" refers to the module on which you are currently working.

–ed (Copy Files to Export Directory)

–ed *export_directory*

The –ed option copies files listed in the Export line of the flow file to the directory you specify. If you do not use the –ed option, the files are copied to the working directory. See "Flow Files" for a description of the Export line of the flow file.

If you use the –ed option with the –wd option and do not specify an absolute path name for the export directory, the export directory is placed underneath the working directory.

In the following example, the export3 directory is created underneath the sub3 directory:

xflow -implement balanced.opt -wd sub3 -ed export3 testclk.vhd

If you do not want the export directory to be a subdirectory of the working directory, enter an absolute path name as in the following example:

xflow -implement balanced.opt -wd sub3 -ed /usr/export3 testclk.vhd

–f (Execute Commands File)

–f *command_file*

The –f option executes the command line arguments in the specified *command_file*. For more information on the –f option, see "–f (Execute Commands File)" in Chapter 1.

–g (Specify a Global Variable)

–g *variable*:*value*

The –g option allows you to assign a value to a variable in a flow or option file. This value is applied globally. The following example shows how to specify a global variable at the command line:

```
xflow -implement balanced -g $simulation_output:time_sim calc
```

Note: If a global variable is specified both on the command line and in a flow file, the command line takes precedence over the flow file.

–log (Specify Log File)

The –log option allows you to specify a log filename at the command line. XFLOW writes the log file to the working directory after each run. By default, the log filename is xflow.log.

–norun (Creates a Script File Only)

By default, XFLOW runs the programs enabled in the flow file. Use the –norun option if you do not want to run the programs but instead want to create a script file (SCR, BAT, or TCL). XFLOW copies the appropriate flow and option files to your working directory and creates a script file based on these files. This is useful if you want to check the programs and options listed in the script file before executing them.

Following is an example:

```
xflow -implement balanced.opt -norun testclk.edf
```

In this example, XFLOW copies the balanced.opt and fpga.flw files to the current directory and creates the following script file:

```
##########################################
# Script file to run the flow
#
##########################################
#
# Command line for ngdbuild
#
ngdbuild -p xc2v250fg256-5 -nt timestamp /home/
xflow_test/testclk.edf testclk.ngd
#
# Command line for map
#
map -o testclk_map.ncd testclk.ngd testclk.pcf
#
# Command line for par
#
par -w -ol 2 -d 0 testclk_map.ncd testclk.ncd
testclk.pcf
#
# Command line for post_par_trce
#
trce -e 3 -o testclk.twr testclk.ncd testclk.pcf
```

–o (Change Output File Name)

–o *output_filename*

This option allows you to change the output file base name. If you do not specify this option, the output file name has the base name as the input file in most cases.

The following example shows how to use the –o option to change the base name of output files from “testclk” to “newname”:

```
xflow -implement balanced.opt -o newname testclk.edf
```

–p (Part Number)

–p *part*

By default (without the –p option), XFLOW searches for the part name in the input design file. If XFLOW finds a part number, it uses that number as the target device for the design. If XFLOW does not find a part number in the design input file, it prints an error message indicating that a part number is missing.

The –p option allows you to specify a device. For a list of valid ways to specify a part, see "–p (Part Number)" in Chapter 1.

For FPGA part types, you must designate a part name with a package name. If you do not, XFLOW halts at MAP and reports that a package needs to be specified. You can use the PARTGen –i option to obtain package names for installed devices. See "–i (Print a List of Devices, Packages, and Speeds)" in Chapter 4 for information.

For CPLD part types, either the part number or the family name can be specified.

The following example show how to use the –p option for a Virtex design:

xflow -p xc2vp4fg256-6 -implement high_effort.opt testclk.edf

Note: If you are running the Modular Design flow and are targeting a part different from the one specified in your source design, you must specify the part type using the –p option *every time* you run the –initial, –module, or –assemble flow type.

–pd (PIMs Directory)

–pd *pim_directory*

The –pd option is used to specify the PIMS directory. The PIMs directory stores implemented module files when using Modular Design.

–rd (Copy Report Files)

–rd *report_directory*

The –rd option copies the report files output during the XFLOW run from the working directory to the specified directory. The original report files are kept intact in the working directory.

You can create the report directory prior to using this option, or specify the name of the report directory and let XFLOW create it for you. If you do not specify an absolute path name for the report directory, XFLOW creates the specified report directory in your working directory. Following is an example in which the report directory (reportdir) is created in the working directory (workdir):

```
xflow -implement balanced.opt -wd workdir -rd reportdir testclk.edf
```

If you do not want the report directory to be a subdirectory of the working directory, enter an absolute path name, as shown in the following example:

```
xflow -implement balanced.opt -wd workdir -rd /usr/reportdir
testclk.edf
```

–wd (Specify a Working Directory)

–wd *working_directory*

The default behavior of XFLOW (without the –wd option) is to use the directory from which you invoked XFLOW as the working directory. The –wd option allows you to specify a different directory as the working directory. XFLOW searches for all flow files, option files, and input files in the working directory. It also runs all subprograms and outputs files in this directory.

Note: If you use the –wd option and want to use a UCF file as one of your input files, you must copy the UCF file into the working directory.

Unless you specify a directory path, the working directory is created in the current directory. For example, if you enter the following command, the directory sub1 is created in the current directory:

```
xflow -fsim generic_verilog.opt -wd sub1 testclk.v
```

You can also enter an absolute path for a working directory as in the following example. You can specify an existing directory or specify a path for XFLOW to create.

```
xflow -fsim generic_verilog.opt -wd /usr/project1 testclk.v
```

Running XFLOW

The following sections describe common ways to use XFLOW.

Using XFLOW Flow Types in Combination

You can combine flow types on the XFLOW command line to run different flows.

The following example shows how to use a combination of flow types to implement a design, create a bitstream for FPGA device configuration, and generate an EDIF timing simulation netlist for an FPGA design named testclk:

xflow -p xc2v250fg256-5 -implement balanced -tsim generic_verilog -config bitgen testclk

The following example shows how to use a combination of flow types to fit a CPLD design and generate a VHDL timing simulation netlist for a CPLD design named main_pcb:

xflow -p xc2c64-4-cp56 -fit balanced -tsim generic_vhdl main_pcb

Running “Smart Flow”

“Smart Flow” automatically detects changes to your input files and runs the flow from the appropriate point. XFLOW detects changes made to design files, flow files, option files, and trigger files. It also detects and reruns aborted flows. To run “Smart Flow,” type the XFLOW syntax *without* specifying an extension for your input design. XFLOW automatically detects which input file to read and starts the flow at the appropriate point.

For example, if you enter the following command and XFLOW detects changes to the calc.edf file, XFLOW runs *all* the programs in the flow and option files. However, if you enter the same command and XFLOW detects changes only to the calc.mfp file generated by the Floorplanner GUI, XFLOW starts the flow with the MAP program.

```
xflow -implement balanced.opt calc
```

Using the SCR, BAT, or TCL File

Every time you run XFLOW, it creates a script file that includes the command line commands of all the programs run. You can use this file for the following:

- Review this file to check which commands were run
- Execute this file instead of running XFLOW

By default, this file is named xflow_script.bat (PC) or xflow_script.scr (UNIX), although you can specify the output script file type by using the **`$scripts_to_generate`** option. To execute the script file, type **`xflow_script.bat`**, **`xflow_script.scr`**, or **`xflow_script.tcl`** at the command line.

If you choose to execute the script file instead of using XFLOW, the features of "Smart XFLOW" are not enabled. For example, XFLOW starts the flow at an appropriate point based on which files have changed, while the script file simply runs every command listed in the file. In addition, the script file does not provide error detection. For example, if an error is encountered during NGDBuild, XFLOW detects the error and terminates the flow, while the script file continues and runs MAP.

Using the XIL_XFLOW_PATH Environment Variable

This environment variable is useful for team-based design. By default, XFLOW looks for all flow and option files in your working directory. However, this variable allows you to store flow and option files in a central location and copy them to your team members' local directories, ensuring consistency. To use this variable, do the following:

1. Modify the flow and option files as necessary.
2. Copy the flow and option files to the central directory, and provide your team members with the directory location.
3. Instruct your team members to type the following from their working directory:

   `set XIL_XFLOW_PATH=`*`name_of_central_directory`*

   When the team member runs XFLOW, XFLOW copies all flow and option files from the central directory to his or her local directory.

   Note: If you alter the files in the central directory and want to repopulate the users' local directories, they must delete their local copies of the flow and option files, set the XIL_FLOW_PATH environment variable, and rerun XFLOW to copy in the updated files.

Chapter 24

Data2MEM

Data2MEM is compatible with the following families:

- Virtex™/-E
- Virtex-II™
- Virtex-II Pro™/X
- Virtex-4™
- Virtex™-5 LX
- Spartan™, Spartan-II™/E
- Spartan-3™, Spartan-3E™, Spartan-3L™

This chapter contains the following sections:

Data2MEM Overview

Data2MEM is a command line executable that transforms CPU execution code, or pure data, into Block RAM initialization records. A library version of the tool is also used by the NGDBuild, BitGen, NetGen, FPGA Editor, and iMPACT tools to incorporate Data2MEM functionality directly into the operation of those tools.

Data2MEM flow types are:

- Flow for software designers that uses Data2MEM as a command line tool to generate updated .bit and .bmm files.
- Flow for hardware designers that integrates Data2MEM with Xilinx implementation tools. (See applicable chapters in this reference guide for more information.)
- Flow that uses Data2MEM as a command line tool to generate behavioral simulation files. (See Chapter 22, "NetGen" for additional information.)

Data2MEM Syntax

Use the following syntax to run Data2MEM from the command line:

data2mem -bm|bd *infile*.[bmm|elf|mem] *[options]*

options can be any number of the command line options listed in the "Data2MEM Options" section of this chapter. Options need not be listed in any order. Separate multiple options with spaces. See the "Data2MEM Options" section of this chapter.

infile specifies the name of the input file. Use the -bm option to specify a .bmm file and the -bd option to specify an .mem or .elf file.

Data2MEM Input and Output Files

Data2MEM uses a number of input and output files. The following figure shows the range of files, and their input and output relationship to Data2MEM. Below is a description of each file type, and how it is consumed or produced by Data2MEM.

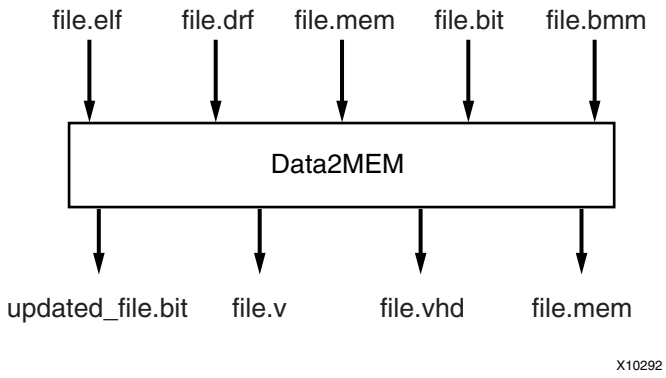


Figure 24-1: **Data2MEM Input and Output Files**

Block RAM Memory Map (.bmm) files

A Block RAM Memory Map (.bmm) file is a simple text file that has a syntactic description of how individual Block RAMs constitute a contiguous logical data space. This is a fundamental input file that Data2MEM uses to direct the translation of data into the proper initialization form. A .bmm file is created primarily manually; however, Data2MEM has facilities to generate .bmm file templates that can be customized to a specific design. A .bmm file can also be created by automated scripting means. Because a .bmm file is a simple text file, it is directly editable. Data2MEM allows the free-form use of both slash (//) and asterisk (/*...*/) commenting styles.

Executable and Linkable Format (.elf) files

An Executable and Linkable Format (.elf) file is a binary data file that contains an executable CPU code image, ready for running on a CPU. ELF files are produced by software compiler tools. Refer to the proper software tool documentation for details on creating .elf files. Data2MEM uses .elf files as its basic data input form. Because .elf files are binary data, they are not directly editable. Data2MEM also provides some facilities for examining the content of .elf files.

Debugging Information Format DWARF (.drf) files

A Debugging Information Format DWARF (.drf) file is a binary data file that contains the executable CPU code image, plus debug information required by symbolic source-level debuggers. These files are produced by the same software compiler tools as .elf files. Data2MEM reads .drf files wherever .elf files can be used. Because .drf files are binary data, they are not directly editable. Data2MEM provides some facilities for examining the content of .drf files.

Memory (.mem) files

A memory (.mem) file is a simple text file that describes contiguous blocks of data. A .mem file may have as many contiguous data blocks as required. There can be any size gap of address range between data blocks; however, no two data blocks can overlap an address range. Because a .mem file is a simple text file, it is directly editable. Data2MEM allows the free-form use of both slash (//) and asterisk (/*...*/) commenting styles.

The format of .mem files is an industry standard, and consists of two basic elements; hex address specifier and hex data values. Data2MEM uses .mem files for both data input and output. See the description of the differences between input and output memory files below.

Memory (.mem) Files as Output

Output memory files are used primarily for Verilog simulations with third-party memory models. Therefore, the format follows industry standard use on three key points:

- All data values must be the same number of bits wide, and must be the same width as expected by the memory model.
- All data values reside within a larger array of values, starting at zero.
- If an address gap exists between two contiguous blocks of data, the data between the gaps still logically exists but is undefined.

Bit (.bit) files

A bitstream (.bit) file is a binary data file that contains a bit image downloadable to an FPGA device. Data2MEM replaces the Block RAM data in .bit files, without the intervention of any Xilinx implementation tools; therefore, Data2MEM both inputs and outputs .bit files. A .bit file is generated by Xilinx implementation tools. Please refer to Xilinx implementation tools in this reference manual for details on creating .bit files. Because .bit files are binary data, they are not directly editable. Data2MEM provides some facilities for examining the content of .bit files.

Verilog (.v) files

A Verilog (.v) file is a simple text file that Data2MEM outputs, which contains “defparm” records to initialize Block RAMs. This file is used primarily for pre-synthesis and post-synthesis simulation. Because a .v file is a simple text file, it is directly editable, but editing this file is not advised because it is a generated file. Data2MEM allows the free-form use of both slash (//) and asterisk (/*...*/) commenting styles.

VHDL (.vhd) files

A VHDL (.vhd) file is a simple text file that Data2MEM outputs, which contains "bit_vector" constants to initialize Block RAMs. These constants can be used in "generic maps" to instance an initialized Block RAM. This file is used primarily for pre-synthesis and post-synthesis simulation. Because a .vhd file is a simple text file, it is directly editable. However, because this file is a generated file, editing is not advised. Data2MEM allows the free-form use of both slash (//) and asterisk (/*...*/) commenting styles.

UCF (.ucf) files

A User Constraints File (.ucf) is a simple text file that Data2MEM outputs, which contains INST records to initialize Block RAMs. Because a .ucf file is a simple text file, it is directly editable. However, because this file is a generated file, editing is not advised. Data2MEM allows the free-form use of both slash (//) and asterisk (/*...*/) commenting styles.

This file type is supported for legacy workflows. Its use for new designs or workflows is discouraged.

Data2MEM Options

The following table lists the Data2MEM command line options:

Table 24-1: **Data2MEM Command Line Options**

Option	Description
–**bd** *[tagname] filename*.**[elf|mem]**	The -bd option specifies the name of the input ELF or MEM file. If the file extension is missing, .elf is assumed. The .mem extension must be supplied to denote a MEM file. If TagNames are given, only the address space of the same names within the BMM file are used for translation. All other input file data outside of the TagName address spaces are ignored. If no further options are specified, "-o u filename" functionality is assumed. One or more –bd options can be specified on the command line.
–**bm** *filename*.**bmm**	The -bm option specifies the name of the input BMM file. If the file extension is missing, a .bmm file extension is assumed. If a filename is not specified, the ELF or MEM root filename with a .bmm extension is assumed. If only this option is given, then Data2MEM checks the syntax of the BMM file and reports any errors. The -bm option can be specified once on the command line.
–**bt** *filename*	The -bt option specifies the name of the input BIT file. If the file extension is missing, .bit is assumed. If the –o option is not specified with this option, the output BIT filename will have the same root filename as the input BIT file, with "_rp" appended at the end. A .bit file extension is assumed.

Table 24-1: **Data2MEM Command Line Options**

Option	Description
–f *filename*	The -f option specifies the name of an option file. If the file extension is missing, an .opt file extension is assumed. Option files are identical to the command line options, but contained in a text file. An option, and its attributes, must appear on the same text line. The -f option can be specified once on the command line, and must not be contained in the .opt file.
–h [*option]* | *support*	The -h option displays a list of supported command line options. When specified with a command line option, a detailed description and use of the option appears. When specified with the *support* argument, a list of supported parts is displayed.
–i	The -i option directs Data2MEM to ignore ELF or MEM data that is outside of the address space that is defined in the BMM file.
–intstyle ise | xflow | silent	The –intstyle option reduces screen output based on the integration style that you are running. When using the –intstyle option, one of three modes must be specified: *ise*, *xflow*, or *silent*. The specified mode sets the way information is displayed in the following ways: **–intstyle ise** This mode shows the program is being run as part of an integrated design environment. **–intstyle xflow** This mode shows the program is being run as part of an integrated batch flow. **–intstyle silent** This mode limits screen output to warning and error messages only.
–**log** *filename*	The -log option directs Data2MEM to generate a log file that contains all enabled Data2MEM messages. If a file extension is not specified, a .dmr file extension is assumed. If a log filename is not specified, the log file assumes the root filename of the input BMM file.

Table 24-1: **Data2MEM Command Line Options**

Option	Description
–**o** u | v | h | m *filename*	The -o option specifies the name of the output files. The string preceding the file name indicates which file formats are to be output. No spaces can separate the specified file types, and file types can appear in any order. One or more file types can be specified. The file type are: 'u' - specifies a UCF file format, with a .ucf file extension 'v' - specifies a Verilog file format, with a .v file extension 'h' - specifies a VHDL file format, with a .vhd file extension 'm' - specifies a MEM file format, with a .mem file extension The *filename* applies to all specified output file types. If the file extension is missing, the appropriate file extension will be added to the output files.
–**p** *partname*	Specifies the name of the target Xilinx part. If unspecified, a XCV50 part is assumed. To view a list of supported parts, use the -h *support* option.
–**pp** *filename*	The -pp option specifies the name of an input preprocess file. If a filename is not specified, a .bmm file extension is assumed. If this option is specified with the -o option, the preprocessed output is sent to the specified output file. Otherwise, preprocessed output is displayed on the console.
–q e | w | i	The -q option reduces the output of Data2MEM messages. Message type arguments are: 'e' - disables ERROR messages 'w' - disables WARNING messages 'i' - disables INFO messages.
–u	The -u option updates the text output files, specified with the -o option, for all address spaces. Depending on the file type, an output file is either empty, or will contain initializations of all zero. If this option is not used, only address spaces that receive transformed data are output.
–w off | on	The -w option prevents or allows file overwrites. Specifying this option with the *off* argument directs Data2MEM to output an error if a file will be overwritten. Using this option with the *on* argument allows Data2MEM to overwrite files without issuing a warning message. By default this option is *on.*

XILINX®

Appendix A

Xilinx Development System Files

This appendix gives an alphabetic listing of the files used by the Xilinx development system.

Name	Type	Produced By	Description
BIT	Data	BitGen	Download bitstream file for devices containing all of the configuration information from the NCD file
BGN	ASCII	BitGen	Report file containing information about a BitGen run
BLD	ASCII	NGDBuild	Report file containing information about an NGDBuild run, including the subprocesses run by NGDBuild
DATA	C File	TRCE	File created with the –stamp option to TRCE that contains timing model information
DC	ASCII	Synopsys FPGA Compiler	Synopsys setup file containing constraints read into the Xilinx Development System
DLY	ASCII	PAR	File containing delay information for each net in a design
DRC	ASCII	BitGen	Design Rule Check file produced by BitGen
EDIF (various file extensions)	ASCII	CAE vendor's EDIF 2 0 0 netlist writer.	EDIF netlist. The Xilinx Development System accepts an EDIF 2 0 0 Level 0 netlist file
EDN	ASCII	NGD2EDIF	Default extension for an EDIF 2 0 0 netlist file
ELF	ASCII	Used for NetGen	This file populates the Block RAMs specified in the .bmm file.

Name	Type	Produced By	Description
EPL	ASCII	FPGA Editor	FPGA Editor command log file. The EPL file keeps a record of all FPGA Editor commands executed and output generated. It is used to recover an aborted FPGA Editor session
EXO	Data	PROMGen	PROM file in Motorola's EXORMAT format
FLW	ASCII	Provided with software	File containing command sequences for XFLOW programs
INI	ASCII	Xilinx software	Script that determines what FPGA Editor commands are performed when the FPGA Editor starts up
GYD	ASCII	CPLDfit	CPLD guide file
HEX	Hex	PROMGen Command	Output file from PROMGEN that contains a hexadecimal representation of a bitstream
IBS	ASCII	IBISWriter Command	Output file from IBISWriter that consists of a list of pins used by the design, the signals internal to the device that connect to those pins, and the IBIS buffer models for the IOBs connected to the pins
JED	JEDEC	CPLDfit	Programming file to be downloaded to a device
LOG	ASCII	XFLOW TRACE	Log file containing all the messages generated during the execution of XFLOW (xflow.log) TRACE (*macro*.log)
LL	ASCII	BitGen	Optional ASCII logic allocation file with a .ll extension. The logic allocation file indicates the bitstream position of latches, flip-flops, and IOB inputs and outputs.
MEM	ASCII	User (with text editor)	User-edited memory file that defines the contents of a ROM
MCS	Data	PROMGen	PROM-formatted file in Intel's MCS-86 format
MDF	ASCII	MAP	A file describing how logic was decomposed when the design was mapped. The MDF file is used for guided mapping.

Name	Type	Produced By	Description
MFP	ASCII	Floorplanner	Map Floorplanner File, which is generated by the Floorplanner, specified as an input file with the –fp option. The MFP file is essentially used as a guide file for mapping.
MOD	ASCII	TRACE	File created with the –stamp option in TRCE that contains timing model information
MRP	ASCII	MAP	MAP report file containing information about a technology mapper command run
MSK	Data	BitGen	File used to compare relevant bit locations when reading back configuration data contained in an operating Xilinx device
NAV	XML	NGDBuild	Report file containing information about an NGDBuild run, including the subprocesses run by NGDBuild. From this file, the user can click any linked net or instance names to navigate back to the net or instance in the source design.
NCD	Data	MAP, PAR, FPGA Editor	Flat physical design database correlated to the physical side of the NGD in order to provide coupling back to the user's original design
NCF	ASCII	CAE Vendor toolset	Vendor-specified logical constraints files
NGA	Data	NetGen	Back-annotated mapped NCD file
NGC	Binary	XST	Netlist file with constraint information.
NGD	Data	NGDBuild	Generic Database file. This file contains a logical description of the design expressed both in terms of the hierarchy used when the design was first created and in terms of lower-level Xilinx primitives to which the hierarchy resolves.
NGM	Data	MAP	File containing all of the data in the input NGD file as well as information on the physical design produced by the mapping. The NGM file is used for back-annotation.

Name	Type	Produced By	Description
NGO	Data	Netlist Readers	File containing a logical description of the design in terms of its original components and hierarchy
NKY	Data	BitGen	Encryption key file
NLF	ASCII	NetGen	NetGen log file that contains information on the NetGen run
NMC	Binary	FPGA Editor	Xilinx physical macro library file containing a physical macro definition that can be instantiated into a design
OPT	ASCII	XFLOW	Options file used by XFLOW
PAD	ASCII	PAR	File containing a listing of all I/O components used in the design and their associated primary pins
PAR	ASCII	PAR	PAR report file containing execution information about the PAR command run. The file shows the steps taken as the program converges on a placement and routing solution
PCF	ASCII	MAP, FPGA Editor	File containing physical constraints specified during design entry (that is, schematics) and constraints added by the user
PIN	ASCII	NetGen	Cadence signal-to-pin mapping file
PRM	ASCII	PROMGen	File containing a memory map of a PROM file showing the starting and ending PROM address for each BIT file loaded
RBT	ASCII	BitGen	“Rawbits" file consisting of ASCII ones and zeros representing the data in the bitstream file
RPT	ASCII	PIN2UCF	Report file generated by PIN2UCF when conflicting constraints are discovered. The name is pinlock.rpt.
RCV	ASCII	FPGA Editor	FPGA Editor recovery file
SCR	ASCII	FPGA Editor or XFLOW	FPGA Editor or XFLOW command script file
SDF	ASCII	NetGen	File containing the timing data for a design. Standard Delay Format File
SVF	ASCII	NetGen	Assertion file written for Formality equivalency checking tool

Name	Type	Produced By	Description
TCL	ASCII	User (with text editor)	TCL script file
TDR	ASCII	DRC	Physical DRC report file
TEK	Data	PROMGen	PROM-formatted file in Tektronix's TEKHEX format
TV	ASCII	NetGen	Verilog test fixture file
TVHD	ASCII	NetGen	VHDL testbench file
TWR	ASCII	TRACE	Timing report file produced by TRACE
TWX	XML	TRACE	Timing report file produced by TRACE. From this file, the user can click any linked net or instance names to navigate back to the net or instance in the source design.
UCF	ASCII	User (with text editor)	User-specified logical constraints file
URF	ASCII	User (with text editor)	User-specified rules file containing information about the acceptable netlist input files, netlist readers, and netlist reader options
V	ASCII	NetGen	Verilog netlist
VHD	ASCII	NetGen	VHDL netlist
VM6	Design	CPLDfit	Output file from the CPLDfit
VXC	ASCII	NetGen	Assertion file written for Conformal-LEC equivalence checking tool
XCT	ASCII	PARTGen	File containing detailed information about architectures and devices
XTF	ASCII	Previous releases of Xilinx Development System	Xilinx netlist format file
XPI	ASCII	PAR	File containing PAR run summary

Appendix B

EDIF2NGD, and NGDBuild

This appendix describes the netlist reader program, EDIF2NGD, and how this program interacts with NGDBuild. The appendix contains the following sections:

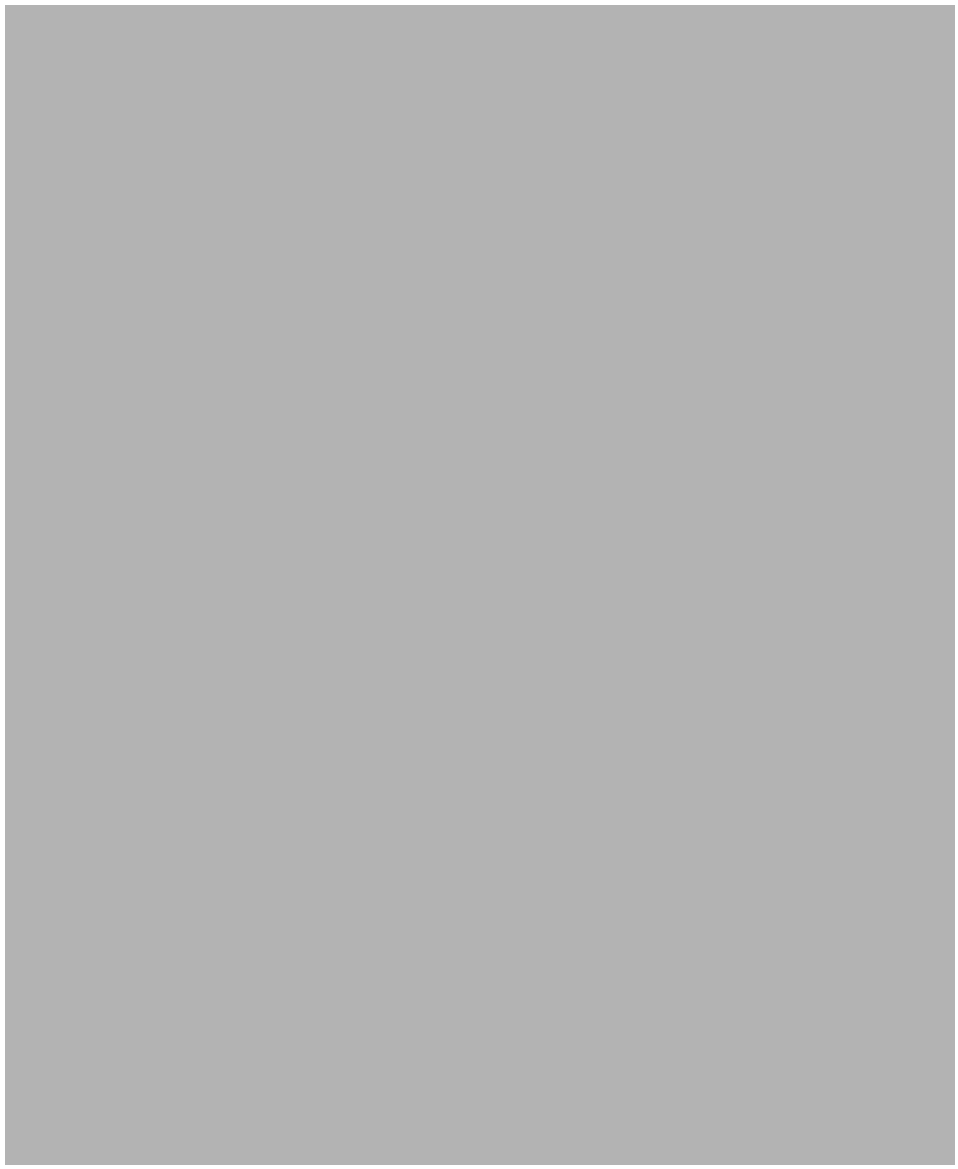
EDIF2NGD

This program is compatible with the following families:

- Virtex™, Virtex™-E
- Virtex™-II
- Virtex™-II Pro, Virtex™-II Pro X
- Virtex™-4
- Virtex™-5 LX
- Spartan™-II, Spartan™-IIE
- Spartan™-3, Spartan™-3E, Spartan™-3L
- CoolRunner™ XPLA3, CoolRunner™-II
- XC9500™, XC9500XL™, XC9500XV™

The EDIF2NGD program allows you to read an EDIF (Electronic Design Interchange Format) 2 0 0 file into the Xilinx development system toolset. EDIF2NGD converts an industry-standard EDIF netlist to an NGO file—a Xilinx-specific format. The EDIF file includes the hierarchy of the input schematic. The output NGO file is a binary database describing the design in terms of the components and hierarchy specified in the input design file. After you convert the EDIF file to an NGO file, you run NGDBuild to create an NGD file, which expands the design to include a description reduced to Xilinx primitives.

The following figure shows the flow through EDIF2NGD.



EDIF2NGD Design Flow

You can run EDIF2NGD in the following ways:

- Automatically from NGDBuild
- From the UNIX or DOS command line, as described in the following sections

Note: When creating net or symbol names, do not use reserved names. Reserved names are the names of symbols for primitives and macros in the *Libraries Guide* and net names GSR, RESET, GR, and PRELOAD. If you use these names, EDIF2NGD issues an error.

EDIF2NGD Syntax

The following command reads your EDIF netlist and converts it to an NGO file:

edif2ngd [*options*] *edif_file ngo_file*

options can be any number of the EDIF2NGD options listed in "EDIF2NGD Options". They do not need to be listed in any particular order. Separate multiple options with spaces.

edif_file is the EDIF 2 0 0 input file to be converted. If you enter a file name with no extension, EDIF2NGD looks for a file with the name you specified and an .edn extension. If the file has an extension other than .edn, you must enter the extension as part of *edif_file*.

Note: For EDIF2NGD to read a Mentor Graphics EDIF file, you must have installed the Mentor Graphics software component on your system. Similarly, to read a Cadence EDIF file, you must have installed the Cadence software component.

ngo_file is the output file in NGO format. The output file name, its extension, and its location are determined in the following ways:

- If you do not specify an output file name, the output file has the same name as the input file, with an .ngo extension.
- If you specify an output file name with no extension, EDIF2NGD appends the .ngo extension to the file name.
- If you specify a file name with an extension other than .ngo, you get an error message and EDIF2NGD does not run.
- If you do not specify a full path name, the output file is placed in the directory from which you ran EDIF2NGD.

If the output file exists, it is overwritten with the new file.

EDIF2NGD Input Files

EDIF2NGD uses the following files as input:

- EDIF file—This is an EDIF 2 0 0 netlist file. The file must be a Level 0 EDIF netlist, as defined in the EDIF 2 0 0 specification. The Xilinx Development System toolset can understand EDIF files developed using components from any of these libraries:
  - Xilinx Unified Libraries (described in the *Libraries Guide*)
  - XSI (Xilinx Synopsys Interface) Libraries
  - Any Xilinx physical macros you create

  Note: Xilinx tools do not recognize Xilinx Unified Libraries components defined as macros; they only recognize the primitives from this library. The third-party EDIF writer must include definitions for all macros.
- NCF file—This Netlist Constraints File is produced by a vendor toolset and contains constraints specified within the toolset. EDIF2NGD reads the constraints in this file and adds the constraints to the output NGO file.

  EDIF2NGD reads the constraints in the NCF file if the NCF file has the same base name as the input EDIF file and an .ncf extension. The name of the NCF file does not have to be entered on the EDIF2NGD command line.

EDIF2NGD Output Files

The output of EDIF2NGD is an NGO file—a binary file containing a logical description of the design in terms of its original components and hierarchy.

EDIF2NGD Options

This section describes the EDIF2NGD command line options.

–a (Add PADs to Top-Level Port Signals)

The –a option adds PAD properties to all top-level port signals. This option is necessary if the EDIF2NGD input is an EDIF file in which PAD symbols were translated into ports. If you do not specify a –a option for one of these EDIF files, the absence of PAD instances in the EDIF file causes EDIF2NGD to read the design incorrectly. Subsequently, MAP interprets the logic as unused and removes it.

In all Mentor Graphics and Cadence EDIF files, PAD symbols are translated into ports. For EDIF files from either of these vendors, the –a option is set automatically; you do not have to enter the –a option on the EDIF2NGD command line.

–aul (Allow Unmatched LOCs)

By default (without the –aul option), EDIF2NGD generates an error if the constraints specified for pin, net, or instance names in the NCF file cannot be found in the design. If this error occurs, an NGO file is not written. If you enter the –aul option, EDIF2NGD generates a warning instead of an error for LOC constraints and writes an NGO file.

You may want to run EDIF2NGD with the –aul option if your constraints file includes location constraints for pin, net, or instance names that have not yet been defined in the HDL or schematic. This allows you to maintain one version of your constraints files for both partially complete and final designs.

Note: When using this option, make sure you do not have misspelled net or instance names in your design. Misspelled names may cause inaccurate placing and routing.

–f (Execute Commands File)

–f *command_file*

The –f option executes the command line arguments in the specified *command_file*. For more information on the –f option, see "–f (Execute Commands File)" in Chapter 1.

–intstyle (Integration Style)

`-intstyle` {**`ise`** | **`xflow`** | **`silent`**}

The –intstyle option reduces screen output based on the integration style you are running. When using the –intstyle option, one of three modes must be specified: *ise*, *xflow*, or *silent*. The mode sets the way information is displayed in the following ways:

`-intstyle ise`

This mode indicates the program is being run as part of an integrated design environment.

`-intstyle xflow`

This mode indicates the program is being run as part of an integrated batch flow.

`-intstyle silent`

This mode limits screen output to warning and error messages only.

Note: The -intstyle option is automatically invoked when running in an integrated environment, such as Project Navigator or XFLOW.

–l (Libraries to Search)

–l *libname*

The –l option specifies a library to search when determining what library components were used to build the design. This information is necessary for NGDBuild, which must determine the source of the design's components before it can resolve the components to Xilinx primitives.

You may specify multiple –l options on the command line. Each must be preceded with –l; you cannot combine multiple *libname* specifiers after one –l. For example, **–l xilinxun synopsys** is not acceptable, while **–l xilinxun –l synopsys** is acceptable.

The allowable entries for *libname* are the following.

- **xilinxun** (For Xilinx Unified library)
- **synopsys**

Note: You do not have to enter xilinxun with a –l option. The Xilinx Development System tools automatically access these libraries. You do not have to enter synopsys with a –l option if the EDIF netlist contains an author construct with the string "Synopsys." In this case, EDIF2NGD automatically detects that the design is from Synopsys.

–p (Part Number)

–p *part*

The –p option specifies the part into which your design is implemented. The –p option can specify an architecture only, a complete part specification (device, package, and speed), or a partial specification (for example, device and package only).

The syntax for the –p option is described in "–p (Part Number)" in Chapter 1. Examples of *part* entries are **XCV50-TQ144** and **XCV50-TQ144-5**.

If you do not specify a part when you run EDIF2NGD, you must specify one when you run NGDBuild.

You can also use the –p option to override a part name in the input EDIF netlist or a part name in an NCF file.

–r (Ignore LOC Constraints)

The –r option filters out all location constraints (LOC=) from the design. If the output file already exists, it is overwritten with the new file.

NGDBuild

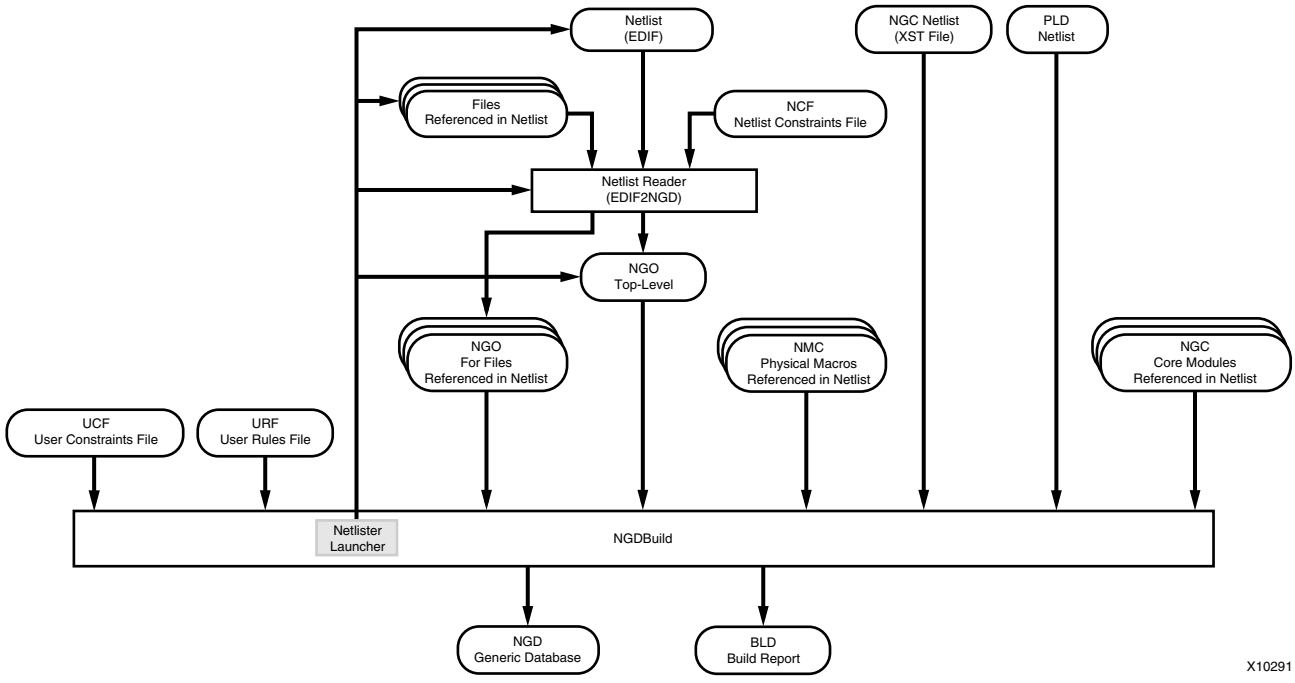
This program is compatible with the following families:

- Virtex™, Virtex™-E
- Virtex™-II
- Virtex™-II Pro, Virtex™-II Pro X
- Virtex™-4
- Virtex™-5 LX
- Spartan™-II, Spartan™-IIE
- Spartan™-3, Spartan™-3E, Spartan™-3L
- CoolRunner™ XPLA3, CoolRunner™-II
- XC9500™, XC9500XL™, XC9500XV™

NGDBuild performs all the steps necessary to read a netlist file in EDIF format and create an NGD file describing the logical design. The NGD file resulting from an NGDBuild run contains both a logical description of the design reduced to NGD primitives and a description in terms of the original hierarchy expressed in the input netlist. The output NGD file can be mapped to the desired device family.

Converting a Netlist to an NGD File

The following figure shows the NGDBuild conversion process.



NGDBuild and the Netlist Readers

NGDBuild performs the following steps to convert a netlist to an NGD file:

1. Reads the source netlist

   To perform this step, NGDBuild invokes the Netlist Launcher (Netlister), a part of the NGDBuild software which determines the type of the input netlist and starts the appropriate netlist reader program. If the input netlist is in EDIF format, the Netlist Launcher invokes EDIF2NGD. If the input netlist is in another format that the Netlist Launcher recognizes, the Netlist Launcher invokes the program necessary to convert the netlist to EDIF format, then invokes EDIF2NGD. The netlist reader produces an NGO file for the top-level netlist file.

   If any subfiles are referenced in the top-level netlist (for example, a PAL description file, or another schematic file), the Netlist Launcher invokes the appropriate netlist reader for each of these files to convert each referenced file to an NGO file.

   The Netlist Launcher is described in "Netlist Launcher (Netlister)". The netlist reader programs are described in "EDIF2NGD".

2. Reduces all components in the design to NGD primitives

   To perform this step, NGDBuild merges components that reference other files by finding the referenced NGO files. NGDBuild also finds the appropriate system library components, physical macros (NMC files) and behavioral models.

3. Checks the design by running a Logical DRC (Design Rule Check) on the converted design

   The Logical DRC is a series of tests on the logical design. It is described in Chapter 5, "Logical Design Rule Check".

4. Writes an NGD file as output

When NGDBuild reads the source netlist, it detects any files or parts of the design that have changed since the last run of NGDBuild. It updates files as follows:

- If you modified your input design, NGDBuild updates all of the files affected by the change and uses the updated files to produce a new NGD file.

  The Netlist Launcher checks timestamps (date and time information) for netlist files and intermediate NGDBuild files (NGOs). If an NGO file has a timestamp earlier than the netlist file that produced it, the NGO file is updated and a new NGD file is produced.

- NGDBuild completes the NGD production if all or some of the intermediate files already exist. These files may exist if you ran a netlist reader before you ran NGDBuild. NGDBuild uses the existing files and creates the remaining files necessary to produce the output NGD file.

Note: If the NGO for an netlist file is up to date, NGDBuild looks for an NCF file with the same base name as the netlist in the netlist directory and compares the timestamp of the NCF file against that of the NGO file. If the NCF file is newer, EDIF2NGD is run again. However, if an NCF file existed on a previous run of NGDBuild and the NCF file was deleted, NGDBuild does not detect that EDIF2NGD must be run again. In this case, you must use the –nt on option to force a rebuild. The –nt on option must also be used to force a rebuild if you change any of the EDIF2NGD options.

Syntax, files, and options for NGDBuild are described in Chapter 6, "NGDBuild".

Bus Matching

When NGDBuild encounters an instance of one netlist within another netlist, it requires that each pin specified on the upper-level instance match to a pin (or port) on the lower-level netlist. Two pins must have exactly the same name in order to be matched. This requirement applies to all FPGAs and CPLDs supported for NGDBuild.

If the interface between the two netlists uses bused pins, these pins are expanded into scalar pins before any pin matching occurs. For example, the pin A[7:0] might be expanded into 8 pins namedA[7] through A[0]. If both netlists use the same nomenclature (that is, the same index delimiter characters) when expanding the bused pin, the scalar pin names will match exactly. However, if the two netlists were created by different vendors and different delimiters are used, the resulting scalar pin names do not match exactly.

In cases where the scalar pin names do not match exactly, NGDBuild analyzes the pin names in both netlists and attempts to identify names that resulted from the expansion of bused pins. When it identifies a bus-expanded pin name, it tries several other bus-naming conventions to find a match in the other netlist so it can merge the two netlists. For example, if it finds a pin named A(3) in one netlist, it looks for pins named A(3), A[3], A<3> or A3 in the other netlist.

The following table lists the bus naming conventions understood by NGDBuild.

Bus Naming Conventions

Naming Convention	Example
busname(index)	DI(3)
busname<index>	DI<3>
busname[index]	DI[3]
busnameindex	DI3

If your third-party netlist writer allows you to specify the bus-naming convention, use one of the conventions shown in the preceding table to avoid “pin mismatch” errors during NGDBuild. If your third-party EDIF writer preserves bus pins using the EDIF “array” construct, the bus pins are expanded by EDIF2NGD using parentheses, which is one of the supported naming conventions.

Note: NGDBuild support for bused pins is limited to this understanding of different naming conventions. It is not able to merge together two netlists if a bused pin has different indices between the two files. For example, it cannot match A[7:0] in one netlist to A[15:8] in another.

In the Xilinx UnifiedPro library for Virtex, some of the pins on the block RAM primitives are bused. If your third-party netlist writer uses one of the bus naming conventions listed in the preceding table or uses the EDIF array construct, these primitives are recognized properly by NGDBuild. The use of any other naming convention may result in an “unexpanded block” error during NGDBuild.

Netlist Launcher (Netlister)

The Netlist Launcher, which is part of NGDBuild, translates an EDIF netlist to an NGO file. NGDBuild uses this NGO file to create an NGD file.

Note: The NGC netlist file does not require Netlist Launcher processing. It is equivalent to an NGO file. Also, it contains its own constraints information and cannot be processed with an NCF file.

When NGDBuild is invoked, the Netlist launcher goes through the following steps:

1. The Netlist Launcher initializes itself with a set of rules for determining what netlist reader to use with each type of netlist, and the options with which each reader is invoked.

   The rules are contained in the system rules file (described in "System Rules File") and in the user rules file (described in "User Rules File").

2. NGDBuild makes the directory of the top-level netlist the first entry in the Netlist Launcher's list of search paths.
3. For the top-level design and for each file referenced in the top-level design, NGDBuild queries the Netlist Launcher for the presence of the corresponding NGO file.
4. For each NGO file requested, the Netlist Launcher performs the following actions:
   - Determines what netlist is the source for the requested NGO file

     The Netlist Launcher determines the source netlist by looking in its rules database for the list of legal netlist extensions. Then, it looks in the search path (which includes the current directory) for a netlist file possessing a legal extension and the same name as the requested NGO file.

   - Finds the requested NGO file

     The Netlist Launcher looks first in the directory specified with the –dd option (or current directory if a directory is not specified). If the NGO file is not found there and the source netlist was not found in the search path, the Netlist Launcher looks for the NGO file in the search path.

   - Determines whether the NGO file must be created or updated

     If neither the netlist source file nor the NGO file is found, NGDBuild exits with an error.

     If the netlist source file is found but the corresponding NGO file is not found, the Netlist Launcher invokes the proper netlist reader to create the NGO file.

     If the netlist source file is not found but the corresponding NGO file is found, the Netlist Launcher indicates to NGDBuild that the file exists and NGDBuild uses this NGO file.

     If both the netlist source file and the corresponding NGO file are found, the netlist file's time stamp is checked against the NGO file's timestamp. If the timestamp of the NGO file is later than the source netlist, the Netlist Launcher returns a "found" status to NGDBuild. If the timestamp of the NGO file is earlier than the netlist source, or the NGO file is not present in the expected location, then the Launcher creates the NGO file from the netlist source by invoking the netlist reader specified by its rules.

   Note: The timestamp check can be overridden by options on the NGDBuild command line. The –nt on option updates all existing NGO files, regardless of their timestamps. The –nt off option does not update any existing NGO files, regardless of their timestamps.

5. The Netlist launcher indicates to NGDBuild that the requested NGO files have been found, and NGDBuild can process all of these NGO files.

Netlist Launcher Rules Files

The behavior of the Netlist Launcher is determined by rules defined in the system rules file and the user rule file. These rules determine the following:

- What netlist source files are acceptable
- Which netlist reader reads each of these netlist files
- What the default options are for each netlist reader

The system rules file contains the default rules supplied with the Xilinx Development System software. The user rules file can add to or override the system rules.

User Rules File

The user rules file can add to or override the rules in the system rules file. You can specify the location of the user rules file with the NGDBuild –ur option. The user rules file must have a .urf extension. See "–ur (Read User Rules File)" in Chapter 6 for more information.

User Rules and System Rules

User rules are treated as follows:

- A user rule can override a system rule if it specifies the same source and target files as the system rule.
- A user rule can supplement a system rule if its target file is identical to a system rule's source file, or if its source file is the same as a system rule's target file.
- A user rule that has a source file identical to a system rule's target file and a target file that is identical to the same system rule's source file is illegal, because it defines a loop.

User Rules Format

Each rule in the user rules file has the following format:

RuleName = <rulename1>;

<key1> = <value1>;

<key2> = <value2>;

.

.

.

<keyn> = <valuen>;

Following are the keys allowed and the values expected:

Note: The value types for the keys are described in "Value Types in Key Statements".

- RuleName—This key identifies the beginning of a rule. It is also used in error messages relating to the rule. It expects a RULENAME value. A value is required.
- NetlistFile—This key specifies a netlist or class of netlists that the netlist reader takes as input. The extension of NetlistFile is used together with the TargetExtension to identify the rule. It expects either a FILENAME or an EXTENSION value. If a file name is specified, it should be just a file name (that is, no path). Any leading path is ignored. A value is required.
- TargetExtension—This key specifies the class of files generated by the netlist reader. It is used together with the extension from NetlistFile to identify the rule. It expects an EXTENSION value. A value is required.
- Netlister—This key specifies the netlist reader to use when translating a specific netlist or class of netlists to a target file. The specific netlist or class of netlists is specified by NetlistFile, and the class of target files is specified by TargetExtension. It expects an EXECUTABLE value. A value is required.
- NetlisterTopOptions—This key specifies options for the netlist reader when compiling the top-level design. It expects an OPTIONS value or the keyword NONE. Included in this string should be the keywords $INFILE and $OUTFILE, in which the input and output files is substituted. In addition, the following keywords may appear.
  - $PART—The part passed to NGDBuild by the –p option is substituted. It may include architecture, device, package and speed information. The syntax for a $PART specification is the same as described in "–p (Part Number)" in Chapter 1.
  - $FAMILY—The family passed to NGDBuild by the –p option is substituted. A value is optional.
  - $DEVICE—The device passed to NGDBuild by the –p option is substituted. A value is optional.
  - $PKG—The package passed to NGDBuild by the –p option is substituted. A value is optional.
  - $SPEED—The speed passed to NGDBuild by the –p option is substituted. A value is optional.
  - $LIBRARIES—The libraries passed to NGDBuild. A value is optional.
  - $IGNORE_LOCS—Substitute the –r option to EDIF2NGD if the NGDBuild command line contained a –r option.
  - $ADD_PADS—Substitute the –a option to EDIF2NGD if the NGDBuild command line contained a –a option.

  The options in the NetlisterTopOptions line must be enclosed in quotation marks.
- NetlisterOptions—This key specifies options for the netlist reader when compiling sub-designs. It expects an OPTIONS value or the keyword NONE. Included in this string should be the keywords $INFILE and $OUTFILE, in which the input and output files is substituted. In addition, any of the keywords that may be entered for the NetlisterTopOptions key may also be used for the NetlisterOptions key.

  The options in the NetlisterOptions line must be enclosed in quotation marks.

- NetlisterDirectory—This key specifies the directory in which to run the netlist reader. The launcher changes to this directory before running the netlist reader. It expects a DIR value or the keywords $SOURCE, $OUTPUT, or NONE, where the path to the source netlist is substituted for $SOURCE, the directory specified with the -dd option is substituted for $OUTPUT, and the current working directory is substituted for NONE. A value is optional.
- NetlisterSuccessStatus—This key specifies the return code that the netlist reader returns if it ran successfully. It expects a NUMBER value or the keyword NONE. The number may be preceded with one of the following: =, <, >, or !. A value is optional.

Value Types in Key Statements

The value types used in the preceding key statements are the following:

- RULENAME—Any series of characters except for a semicolon ";" and white space (for example, space, tab, newline).
- EXTENSION—A "." followed by an extension that conforms to the requirements of the platform.
- FILENAME—A file name that conforms to the requirements of the platform.
- EXECUTABLE—An executable name that conforms to the requirements of the platform. It may be a full path to an executable or just an executable name. If it is just a name, then the $PATH environment variable is used to locate the executable.
- DIR—A directory name that conforms to the requirements of the platform.
- OPTIONS—Any valid string of options for the executable.
- NUMBER—Any series of digits.
- STRING—Any series of characters in double quotes.

System Rules File

The system rules are shown following. The system rules file is not an ASCII file, but for the purpose of describing the rules, the rules are described using the same syntax as in the user rules file. This syntax is described in "User Rules File".

Note: If a rule attribute is not specified, it is assumed to have the value NONE.

```
#########################################################
# edif2ngd rules
#########################################################

RuleName = EDN_RULE;
NetlistFile = .edn;
TargetExtension = .ngo;
Netlister = edif2ngd;
NetlisterTopOptions = "[$IGNORE_LOCS] [$ADD_PADS] [$QUIET] [$AUL] {-l
$LIBRARIES} $INFILE $OUTFILE";
NetlisterOptions = "-noa [$IGNORE_LOCS] {-l $LIBRARIES} $INFILE
$OUTFILE";
NetlisterDirectory = NONE;
NetlisterSuccessStatus = 0;
```

```
RuleName = EDF_RULE;
NetlistFile = .edf;
TargetExtension = .ngo;
Netlister = edif2ngd;
NetlisterTopOptions = "[$IGNORE_LOCS] [$ADD_PADS] [$QUIET] [$AUL] {-l
$LIBRARIES} $INFILE $OUTFILE";
NetlisterOptions = "-noa [$IGNORE_LOCS] {-l $LIBRARIES} $INFILE
$OUTFILE";
NetlisterDirectory = NONE;
NetlisterSuccessStatus = 0;

RuleName = EDIF_RULE;
NetlistFile = .edif;
TargetExtension = .ngo;
Netlister = edif2ngd;
NetlisterTopOptions = "[$IGNORE_LOCS] [$ADD_PADS] [$QUIET] [$AUL] {-l
$LIBRARIES} $INFILE $OUTFILE";
NetlisterOptions = "-noa [$IGNORE_LOCS] {-l $LIBRARIES} $INFILE
$OUTFILE";
NetlisterDirectory = NONE;
NetlisterSuccessStatus = 0;

RuleName = SYN_EDIF_RULE;
NetlistFile = .sedif;
TargetExtension = .ngo;
Netlister = edif2ngd;
NetlisterTopOptions = NONE;
NetlisterOptions = "-l synopsys [$IGNORE_LOCS] {-l $LIBRARIES} $INFILE
$OUTFILE";
NetlisterDirectory = NONE;
NetlisterSuccessStatus = 0;
```

Rules File Examples

The following sections provide examples of system and user rules. The first example is the basis for understanding the ensuing user rules examples.

Example 1: EDF_RULE System Rule

As shown in the "System Rules File", the EDF_RULE system rule is defined as follows.

```
RuleName = EDF_RULE;
NetlistFile = .edf;
TargetExtension = .ngo;
Netlister = edif2ngd;
NetlisterTopOptions = "[$IGNORE_LOCS] [$ADD_PADS] [$QUIET] [$AUL] {-l
$LIBRARIES} $INFILE $OUTFILE";
NetlisterOptions = "-noa [$IGNORE_LOCS] {-l $LIBRARIES} $INFILE
$OUTFILE";
NetlisterDirectory = NONE;
NetlisterSuccessStatus = 0;
```

The EDF_RULE instructs the Netlist Launcher to use EDIF2NGD to translate an EDIF file to an NGO file. If the top-level netlist is being translated, the options defined in NetlisterTopOptions are used; if a lower-level netlist is being processed, the options defined by NetlisterOptions are used. Because NetlisterDirectory is NONE, the Netlist Launcher runs EDIF2NGD in the current working directory (the one from which NGDBuild was launched). The launcher expects EDIF2NGD to issue a return code of 0 if it was successful; any other value is interpreted as failure.

Example 2: User Rule

Following is a another example of a User Rule:

```
// URF Example 2
RuleName = OTHER_RULE; // end-of-line comments are also allowed
NetlistFile = .oth;
TargetExtension = .edf;
Netlister = other2edf;
NetlisterOptions = "$INFILE $OUTFILE";
NetlisterSuccessStatus = 1;
```

The user rule OTHER_RULE defines a completely new translation, from a hypothetical OTH file to an EDIF file. To do this translation, the other2edf program is used. The options defined by NetlisterOptions are used for translating all OTH files, regardless of whether they are top-level or lower-level netlists (because no explicit NetlisterTopOptions is given). The launcher expects other2edf to issue a return code of 1 if it was successful; any other value be interpreted as failure.

After the Netlist Launcher uses OTHER_RULE to run other2edf and create an EDIF file, it uses the EDF_RULE system rule (shown in the preceding section) to translate the EDIF file to an NGO file.

Example 3: User Rule

Following is a another example of a User Rule:

```
// URF Example 3
RuleName = EDF_LIB_RULE;
NetlistFile = .edf;
TargetExtension = .ngo;
NetlisterOptions = "-l xilinxun $INFILE $OUTFILE";
```

Because both the NetlistFile and TargetExtension of this user rule match those of the system rule EDF_RULE (shown in "Example 1: EDF_RULE System Rule"), the EDF_LIB_RULE overrides the EDF_RULE system rule. Any settings that are not defined by the EDF_LIB_RULE are inherited from EDF_RULE. So EDF_LIB_RULE uses the same netlister (EDIF2NGD), the same top-level options, the same directory, and expects the same success status as EDF_RULE. However, when translating lower-level netlists, the options used are only "–l xilinxun $INFILE $OUTFILE." (There is no reason to use "–l xilinxun" on EDIF2NGD; this is for illustrative purposes only.)

Example 4: User Rule

Following is a another example of a User Rule:

```
// URF Example 4
RuleName = STATE_EDF_RULE;
NetlistFile = state.edf;
TargetExtension = .ngo;
Netlister = state2ngd;
```

Although the NetlistFile is a complete file name, this user rule also matches the system rule EDF_RULE (shown in "Example 1: EDF_RULE System Rule"), because the extensions of NetlistFile and TargetExtension match. When the Netlist Launcher tries to make a file called state.ngo, it uses this rule instead of the system rule EDF_RULE (assuming that state.edf exists). As with the previous example, the unspecified settings are inherited from the matching system rule. The only change is that the fictitious program state2ngd is used in place of EDIF2NGD.

Note that if EDF_LIB_RULE (from the example in "Example 3: User Rule") and this rule were both in the user rules file, STATE_EDF_RULE includes the modifications made by EDF_LIB_RULE. So a lower-level state.edf is translated by running state2ngd with the "-l xilinxun" option.

NGDBuild File Names and Locations

Following are some notes about file names in NGDBuild:

- An intermediate file has the same root name as the design that produced it. An intermediate file is generated when more than one netlist reader is needed to translate a netlist to a NGO file.
- Netlist root file names in the search path must be unique. For example, if you have the design state.edn, you cannot have another design named state in any of the directories specified in the search path.
- NGDBuild and the Netlist Launcher support quoted file names. Quoted file names may have special characters (for example, a space) that are not normally allowed.
- If the output directory specified in the call to NGDBuild is not writable, an error is displayed and NGDBuild fails.

Glossary

Click on a letter, or scroll down to view the entire glossary.

A B C D E F G H I J L M N O P R S T U V W

A

ABEL

Advanced Boolean Expression Lanaguage (ABEL) is a high-level language (HDL) and compilation system produced by Data I/O Corporation.

adder

An adder is a combinatorial circuit that computes the sum of two or more numbers.

address

An address is the identification of a storage location, such as a register or a memory cell.

checked for syntax errors.

architecture

Architecture is the common logic structure of a family of programmable integrated circuits. The same architecture can be realized in different manufacturing processes. Examples of Xilinx architectures are the XC9500 devices.

anno

Anno is used to refer to the general back-annotation process.

area constraints

Area constraints are created by the user or a process such as synthesis to direct the optimization process that takes place during design implementation.

ASIC

Application-specific integrated circuit (ASIC), is a full-custom circuit. In which every mask is defined by the customer or a semi-custom circuit (gate array) where only a few masks are defined.

attributes

Attributes are instructions placed on symbols or nets in an FPGA or CPLD schematic to indicate their placement, implementation, naming, directionality, or other properties.I

B

back-annotation

Back-annotation is the translation of a routed or fitted design to a timing simulation netlist.

behavioral design

Behavioral design is a technology-independent, text-based design that incorporates high-level functionality and high-level information flow.

behavioral design method

A behavioral design method defines a circuit in terms of a textual language rather than a schematic of interconnected symbols.

behavioral simulation

Also known as functional simulation. Behavioral simulation is usually performed on designs that are entered using a hardware definition language (HDL).

This type of simulation takes place during the pre-synthesis stage of HDL design. Functional simulation checks that the HDL code describes the desired design behavior.

Behavioral simulation is a simulation process that is performed by interpreting the equations that define the design. The equations do not need to be converted to the logic that represents them.

binary

Binary is a numbering system based on base 2 with only two digits, 0 and 1.

Unsigned binary refers to non-negative binary representation.

bit

A bit is a a binary digit representing 0 or 1.

BIT file

A BIT file is the same as a bitstream file. See bitstream.

BitGen

Is a program that produces a bitstream for Xilinx device configuration. BitGen takes a fully routed NCD (Circuit Description) file as its input and produces a configuration bitstream, a binary file with a.bit extension.

bitstream

A bitstream is a stream of data that contains location information for logic on a device, that is, the placement of Configurable Logic Blocks (CLBs), Input/Output Blocks (IOBs), (TBUFs), pins, and routing elements. The bitstream also includes empty placeholders that are filled with the logical states sent by the device during a readback. Only the memory elements, such as flip-flops, RAMs, and CLB outputs, are mapped to these placeholders, because their contents are likely to change from one state to another. When downloaded to a device, a bitstream configures the logic of a device and programs the device so that the states of that device can be read back.

A bitstream file has a .bit extension.

block

A block is a group of one or more logic functions.

A block is a schematic or symbol sheet. There are four types of blocks.

-- A Composite block indicates that the design is hierarchical.

-- A Module block is a symbol with no underlying schematic.

-- A Pin block represents a schematic pin.

-- An Annotate block is a symbol without electrical connectivity that is used only for documentation and graphics.

bonded

Bonded means connected by a wire.

boundary scan

Boundary scan is the method used for board-level testing of electronic assemblies. The primary objectives are the testing of chip I/O signals and the interconnections between ICs.

It is the method for observing and controlling all new chip I/O signals through a standard interface called a Test Access Port (TAP). The boundary scan architecture includes four dedicated I/O pins for control and is described in IEEE spec 1149.1.

BSDLAnno

A program that automatically modifies a BSDL file for post-configuration interconnect testing.

buffer

A buffer is an element used to increase the current or drive of a weak signal and, consequently, increase the fanout of the signal. A storage element.

BUFT

A BUFT is a 3-state buffer.

bus

A group of two or more signals that carry closely-associated signals in an electronic design.

byte

A binary word consisting of eight bits. When used to store a number value, a byte can represent a number from 0 to 255.

C

CAE

Computer Aided Engineering. The original term for electronic design automation (EDA). Now, often refers to the sofware tools used to develop the manufacturing tooling for the production of electronic system such as for the panelization of circuit boards.

CAE tool

A Computer-Aided Engineering tool (CAE). Usually refers to programs such as Innoveda, Cadence, or Mentor Graphics that are used to perform design entry and design verification.

capacitance

Capacitance is the property that measures the storage of electrically separated charges.

It is also the load on a net.

carry path

The carry path is the computation of the carries in addition or subtraction from one CLB to another.

cell

A cell is a hierarchical description of an FPGA device.

checksum

A checksum is a summation of bits or digits generated according to an arbitrary formula used for checking data integrity. To verify that the data represented by a checksum number has been entered correctly, verify that the checksum number generated after processing is the same as the initial number.

CLB

The Configurable Logic Block (CLB). Constitutes the basic FPGA cell. It includes two 16-bit function generators (F or G), one 8-bit function generator (H), two registers (flip-flops or latches), and reprogrammable routing controls (multiplexers).

CLBs are used to implement macros and other designed functions. They provide the physical support for an implemented and downloaded design. CLBs have inputs on each side, and this versatility makes them flexible for the mapping and partitioning of logic.

CCLK pin

The CCLK pin is the XChecker pin that provides the configuration clock for the device or devices during a download.

clock

A clock is a signal that represents the time that a wave stays at a High or Low state. The rising and falling edges of a clock square wave trigger the activity of the circuits.

clock buffer

A clock buffer is an element used to increase the current or drive of a weak clock signal and consequently increase its fanout.

clock enable

A clock enable is a binary signal that allows or disallows synchronous logic to change with a clock signal. When enabled, this control signal permits a device to be clocked and to become active. There are four different states. The two active High states are CE 0 disabled and CE 1 enabled. The two active Low states are $\overline{CE}$ 0 enabled and $\overline{CE}$ 1 disabled.

clock skew

Clock skew is the time differential between 2 or more destination pins in a path.

CMOS

Complementary Metal Oxide Semiconductor (CMOS). Is an advanced IC manufacturing process technology characterized by high integration, low cost, low power, and high performance.

combinatorial logic

Combinatorial logic refers to any primitives with the exception of storage elements such as flip-flops.

compiler

A compiler is a language interpreter. The Synopsys compiler interprets HDL and makes concurrent process implementations for target architectures.

component

A component is an instantiation or symbol reference from a library of logic elements that can be placed on a schematic.

configuration

Configuration is the process of loading design-specific bitstreams into one or more FPGA devices to define the functional operation of the logical blocks, their interconnections, and the chip I/O.

This concept also refers to the configuration of a design directory for a particular design library.

constraints

Constraints are specifications for the implementation process. There are several categories of constraints: routing, timing, area, mapping, and placement constraints.

Using attributes, you can force the placement of logic (macros) in CLBs, the location of CLBs on the chip, and the maximum delay between flip-flops. PAR does not attempt to change the location of constrained logic.

constraints file

A constraints file specifies constraints (location and path delay) information in a textual form. An alternate method is to place constraints on a schematic.

contention

Contention is the state in which multiple conflicting outputs drive the same net.

counter

A counter is a circuit, composed of registers, that counts pulses, often reacting or causing a reaction to a predetermined pulse or series of pulses. Also called a divider, sometimes accumulator.

CPLD

Complex Programmable Logic Device (CPLD). Is an erasable programmable logic device that can be programmed with a schematic or a behavioral design. CPLDs constitute a type of complex PLD based on EPROM or EEPROM technology. They are characterized by an architecture offering high speed, predictable timing, and simple software.

The basic CPLD cell is called a macrocell, which is the CPLD implementation of a CLB. It is composed of AND gate arrays and is surrounded by the interconnect area.

CPLDs consume more power than FPGA devices, are based on a different architecture, and are primarily used to support behavioral designs and to implement complex counters, complex state machines, arithmetic operations, wide inputs, and PAL crunchers.

CPLDfit

A program that reads in an NGD file and fits the design into the selected CPLD architecture.

D

daisy chain

A daisy chain is a series of bitstream files concatenated in one file. It can be used to program several FPGAs connected in a daisy chain board configuration.

dangling bus

A dangling bus connects to a component pin or net at one end and unconnects at the other. A small filled box at the end of the bus indicates a dangling bus.

dangling net

A dangling net connects to a component pin or net at one end and unconnects at the other. A small filled box at the end of the net indicates a dangling net.

Data2MEM

A program that transforms CPU execution code, or pure data, into Block RAM initialization records.

debugging

Debugging is the process of reading back or probing the states of a configured device to ensure that the device is behaving as expected while in circuit.

decimal

Decimal refers to a numbering system with a base of 10 digits starting with zero.

decoder

A decoder is a symbol that translates *n* input lines of binary information into 2*n* output lines. It is the opposite of an encoder.

Delay Locked Loop (DLL)

A digital circuit used to perform clock management functions on and off-chip.

density

Density is the number of gates on a device.

design implementation

Design implementation is a design implementation specification as opposed to the functional specification of the design. The implementation specification refers to the actual implementation of the design from low-level components expressed in bits. The functional specification refers to the definition of the design or circuit function.

device

A device is an integrated circuit or other solid-state circuit formed in semiconducting materials during manufacturing.

digital

Digital refers to the representation of information by code of discrete elements, as opposed to the continuous scale of analog representation.

DONE pin

The DONE pin is a dual-function pin. As an input, it can be configured to delay the global logic initialization or the enabling of outputs. As an output, it indicates the completion of the configuration process.

downloading

Downloading is the process of configuring or programming a device by sending bitstream data to the device.

DRC

The Design Rule Checker (DRC). A program that checks the (NCD) file for design implementations for errors.

DSP

Digital Signal Processing (DSP). A powerful and flexible technique of processing analog (linear) signals in digital form used in CoreGen.

E

EDA

Electronic Design Automation (EDA). A generic name for all methods of entering and processing digital and analog designs for further processing, simulation, and implementation.

edge decoder

An edge decoder is a decoder whose placement is constrained to precise positions within a side of the FPGA device.

EDIF

EDIF is the Electronic Data Interchange Format, an industry standard file format for specifying a design netlist. It is generated by a third-party design-entry tool. In the Xilinx M1 flow, EDIF is the standard input format.

effort level

Effort level refers to how hard the Xilinx Design System (XDS) tries to place a design. The effort level settings are.

High, which provides the highest quality placement but requires the longest execution time. Use high effort on designs that do not route or do not meet your performance requirements.

Medium, which is the default effort level. It provides the best trade-off between execution time and high quality placement for most designs.

Low, which provides the fastest execution time and adequate placement results for prototyping of simple, easy-to-route designs. Low effort is useful if you are exploring a large design space and only need estimates of final performance.

ENRead

Mentor Graphics EDIF netlist reader. Translates an EDIF netlist into an EDDM single object.

entity

An entity is a set of interconnected components.

EPROM

An EPROM is an erasable PROM, which can be reprogrammed many times. Previous programs are simply erased by exposing the chip to ultra-violet light.

An EEPROM, or electrically erasable PROM, is another variety of EPROM that can be erased electrically.

F

FD

FD is a D flip-flop used in CLBs. Contrast with IFD.

FDSD

FDSD is a D flip-flop with Set Direct.

FIFO

A FIFO is a serial-in/serial-out shift register.

fitting

Fitting is the process of putting logic from your design into physical macrocell locations in the CPLD. Routing is performed automatically, and because of the UIM architecture, all designs are routable.

fitter

The fitter is the software that maps a PLD logic description into the target CPLD.

flat design

A flat design is a design composed of multiple sheets at the top-level schematic.

flattening

Flattening is the process of resolving all of the hierarchy references in a design. If a design contains several instantiations of a logic module, the flattened version of that design will duplicate the logic for each instantiation. A flattened design still contains hierarchical names for instances and nets.

flip-flop

A flip-flop is a simple two-state logic buffer activated by a clock and fed by a single input working in combination with the clock. The states are High and Low. When the clock goes High, the flip-flop works as a buffer as it outputs the value of the D input at the time the clock rises.

The value is kept until the next clock cycle (rising clock edge). The output is not affected when the clock goes Low (falling clock edge).

floorplanning

Floorplanning is the process of choosing the best grouping and connectivity of logic in a design.

It is also the process of manually placing blocks of logic in an FPGA where the goal is to increase density, routability, or performance.

flow

The flow is an ordered sequence of processes that are executed to produce an implementation of a design.

FMAP

An FMAP is a symbol that defines mapping into a 4-input function generator (F or G).

FPGA

Field Programmable Gate Array (FPGA), is a class of integrated circuits pioneered by Xilinx in which the logic function is defined by the customer using Xilinx development system software after the IC has been manufactured and delivered to the end user. Gate arrays are another type of IC whose logic is defined during the manufacturing process. Xilinx supplies RAM-based FPGA devices.

FPGA applications include fast counters, fast pipelined designs, register intensive designs, and battery powered multi-level logic.

function generator

A function generator is a look-up table or black box with three or four inputs implementing any combinational functions of $(2^2)^4$ or 256 functions or $(2^2)^2$ or 65556 functions. The output is any value resulting from the logical functions executed within the box. The function generator implements a complete truth table, allowing speedy prediction of the output.

functional simulation

Functional simulation is the process of identifying logic errors in your design before it is implemented in a Xilinx device. Because timing information for the design is not available, the simulator tests the logic in the design using unit delays. Functional simulation is usually performed at the early stages of the design process.

G

gate

A gate is an integrated circuit composed of several transistors and capable of representing any primitive logic state, such as AND, OR, XOR, or NOT inversion conditions. Gates are also called digital, switching, or logic circuits.

gate array

A gate array is part of the ASIC chip. A gate array represents a certain type of gate repeated all over a VLSI-type chip. This type of logic requires the use of masks to program the connections between the blocks of gates.

global buffers

Global buffers are low-skew, high-speed buffers that connect to long lines. They do not map logic.

There is one BUFGP and one BUFGS in each corner of the chip. Primary buffers must be driven by an IOB. Secondary buffers can be driven by internal logic or IOBs.

global Set/Reset net

A global Set/Reset net is a high-speed, no-skew dedicated net, which reduces delays and routing congestion. This net accesses all flip-flops on the chip and can reinitialize all CLBs and IOBs.

global 3-state net

A global 3-state net is a net that forces all device outputs to high-impedance state unless boundary scan is enabled and executes an EXTEST instruction.

GND pin

The GND pin is Ground (0 volts).

group

A group is a collection of common signals to form a bus. In the case of a counter, for example, the different signals that produce the actual counter values can be combined to form an alias, or group.

guide file

A guide file is a previously placed and routed NCP file that can be used in a subsequent place and route operation.

guided design

Guided design is the use of a previously implemented version of a file for design mapping, placement, and routing. Guided design allows logic to be modified or added to a design while preserving the layout and performance that have been previously achieved.

H

HDL

Hardware Description Language. A language that describes circuits in textual code. The two most widely accepted HDLs are VHDL and Verilog.

An HDL, or hardware description language, describes designs in a technology-independent manner using a high level of abstraction. The most common HDLs in use today are Verilog and VHDL.

hexadecimal

Hexadecimal is a numbering system with a base of 16 digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)

hierarchical annotation

Hierarchical annotation is an advance method of running logical annotation that requires three things: 1) .ncd file, 2) .ngm file, and 3) KEEP_HIERARCHY constraint placed on explicit hierarchy blocks prior to mapping. When this is done, the back-annotation guarantees to preserve the user's hierarchy.

hierarchical design

A hierarchical design is a design composed of multiple sheets at different levels of your schematic.

hold time

Hold time is the time following a clock event during which the data input to a latch or flip-flop must remain stable in order to guarantee that the latched data is correct.

Hprep6

A program that takes an implemented CPLD design (VM6) file from CPLDfit and generates a JEDEC (JED) programming file.

I

IBUF

An IBUF acts as a protection for the chip, shielding it from eventual current overflows.

IC

Integrated Circuit (IC) is a single piece of silicon on which thousands or millions of transistors are combined. ICs are the major building blocks of modern electronic systems.

IEEE

Institute of Electrical and Electronics Engineers.Pronounced I triple E.

IFD

IFD is an IOB flip-flop.

impedance

Impedance is the sum of all resistance and reactance of a circuit to the flow of alternating current.

implementation

Implementation is the mapping, placement and routing of a design. A phase in the design process during which the design is placed and routed.

incremental design

Incremental design refers to the implementation and verification of a small design change using a guide file.

indexes

Indexes are the left-most and right-most bits of a bus defining the bus range and precision.

inertial delay

If the pulse width of a signal is smaller than the path delay (from an input port to an output port) then an inertial delay does not propogate the pulse event through to the output port. This is known as pulse swallowing.

input

An input is the symbol port through which data is sourced.

input pad registers and latches

Input pad registers and latches are D-type registers located in the I/O pad sections of the device. Input pad registers can be used instead of macrocell resources.

instance

An instance is one specific gate or hierarchical element in a design or netlist. The term "symbol" often describes instances in a schematic drawing. Instances are interconnected by pins and nets. Pins are ports through which connections are made from an instance to a net. A design that is flattened to its lowest level constituents is described with primitive instances.

instantiation

Instantiation is the act of placing a symbol that represents a primitive or a macro in a design or netlist.

Integrated Synthesis Environment (ISE)

ISE is an integrated tool suite that enables you to produce, test, and implement designs for Xilinx FPGAs or CPLDs.

interconnect

Interconnect is the metal in a device that is used to implement the nets of the design.

interconnect line

An interconnect line is any portion of a net.

IOB (input/output block)

An IOB is a collection or grouping of basic elements that implement the input and output functions of an FPGA device.

I/O pads

I/O pads are the input/output pads that interface the design logic with the pins of the device.

J

JEDEC

JEDEC is a CPLD file format used for downloading device bitmap information to a device programmer.

L

latch

A latch is a two-state buffer fed by two inputs, D and L. When the L input is Low, it acts as a transparent input; in this case, the latch acts as a buffer and outputs the value input by D. When the L input is High, it ignores the D input value.

library

A library is a set of macros, such as adders, buffers, and flip-flops that is part of the Xilinx interface. A library is used to create schematic designs.

load

A load is an input port.

logic

Logic is one of the three major classes of ICs in most digital electronic systems: microprocessors, memory, and logic. Logic is used for data manipulation and control functions that require higher speed than a microprocessor can provide.

logical annotation

The method of back-annotation that uses the old .ngm and .ncd to attempt to back-annotate the physical information (physical delays) back into the logical (.ngd) file.

logic allocation file

A logic allocation file, design.ll file, designates a file used for probing. The file provides bit locations of the values of RAM, I/O, latches, and flip-flops.

logic optimization

Logic optimization is the process that decreases the area or increases the speed of a design.

longline

A longline connects to a primary global net or to any secondary global net. Each CLB has four dedicated vertical longlines. These lines are very fast.

look-up table (LUT)

A Look-Up Table (LUT), implements Boolean functions.

See function generator.

low

Low is a logical state for which no output is generated.

LSB

An LSB, or least significant bit, is the left-most bit of the bus bounds or indexes. In one-hot and twos-complement encoding, the LSB is the right-most bit.

M

macro

A macro is a component made of nets and primitives, flip-flops, or latches that implements high-level functions, such as adders, subtracters, and dividers. Soft macros and Relationally Placed Macros (RPMs) are types of macros.

macrocell

A macrocell is the CPLD logic cell, which is made of gates only. A macrocell can implement both combinational and registered equations. High-density function block macrocells also contain an arithmetic logic unit (ALU) for implementing arithmetic functions.

mapping

Mapping is the process of assigning a design's logic elements to the specific physical elements that actually implement logic functions in a device.

MCS-86 (Intel)

MCS-86 (Intel) is a PROM format supported by the Xilinx tools. Its maximum address is 1 048 576. This format supports PROM files of up to (8 x 1 048 576) = 8 388 608 bits.

microprocessor

A silicon chip that contains a CPU. Microprocessors control the logic of almost all digital devices, e.g. PCs, workstations, clock radios, and fuel-injection systems for automobiles.

modular design

Modular design refers to the parallel development of pieces, or "modules," of a design being independently worked on and then later merged into one FPGA design.

MSB

The Most Significant Bit (MSB) is the right-most bit of the bus bounds or indexes. In one-hot binary and twos-complement encoding, the MSB is the left-most bit.

MultiLINX

A cable designed to function as a download, read back, verification and logic probing tool for the larger Xilinx devices. MultiLINX functions as a USB device to send and receive data from host.

multiplexer

A multiplexer is a reprogrammable routing control. This component selects one input wire as output from a selection of wires.

N

NCD

A Native Circuit Description (NCD). The physical database format for Xilinx Implementation software tools.

net

A logical connection between two or more symbol instance pins. After routing, the abstract concept of a net is transformed to a physical connection called a wire.

An electrical connection between components or nets. It can also be a connection from a single component. It is the same as a wire or a signal.

netlist

A netlist is a text description of the circuit connectivity. It is basically a list of connectors, a list of instances, and, for each instance, a list of the signals connected to the instance terminals. In addition, the netlist contains attribute information.

network

A network is a collection of logic elements and the wires (nets or connections) that define how they interconnect.

NetGen

A program that reads in applicable Xilinx implementation files, extracts design data, and generates netlists that are used with supported third-party simulation, equivalence checking, and static timing analysis tools.

NGDBuild

A program that converts all input design netlists and then writes the results into a single merged file.

NGM

A design file produced by MAP that contains information about the logical design and information about how the logical design corresponds to the physical design.

O

optimization

Optimization is the process that decreases the area or increases the speed of a design.

oscillator

An oscillator is a bi-stable circuit that can be used as a clock. The stable states are 0 and 1.

P

package

A package is the physical packaging of a chip, for example, PG84, VQ100, and PC48.

pad

A pad is the physical bonding pad on an integrated circuit. All signals on a chip must enter and leave by way of a pad. Pads are connected to package pins in order for signals to enter or leave an integrated circuit package.

PAL

A PAL is a programmable logic device that consists of a programmable AND matrix whose outputs drive fixed OR gates. This was one of the earliest forms of programmable logic. PALs can typically implement small functions easily (up to a hundred gates) and run very fast, but they are inefficient for large functions.

Parallel Cable III

Parallel Cable III is a cable assembly which contains a buffer to protect your PCs parallel port and a set of headers to connect to your target system.

partial reconfiguration

Partial Reconfiguration refers to a portion of an FPGA design being reconfigured while the remainder of the design is still operational.

Partition

A Partition is design unit placed on an instance in a design to enable design preservation.

path

A path is a connected series of nets and logic elements. A path has a start point and an end point that are different depending on the type of path. The time taken for a signal to propagate through a path is referred to as the path delay.

path delay

Path delay is the time it takes for a signal to propagate through a path.

period

The period is the number of steps in a clock pattern multiplied by the step size.

physical annotation

Physical annotation uses just the .ncd file. In this mode, a timing simulation model is generated from the physical device components.

PIM

Physically Implemented Module

pin

A pin can be a symbol pin or a package pin. A package pin is a physical connector on an integrated circuit package that carries signals into and out of an integrated circuit. A symbol pin, also referred to as an instance pin, is the connection point of an instance to a net.

PIP (programmable interconnect points)

Programmable interconnect points, or PIP, provide the routing paths used to connect the inputs and outputs of IOBs and CLBs into logic networks.

A PIP is made of a CMOS transistor, which you can turn on and off to activate the PIP.

placing

Placing is the process of assigning physical device cell locations to the logic in a design.

PLD

A Programmable Logic Device (PLD), is composed of two types of gate arrays: the AND array and the OR array, thus providing for sum of products algorithmic representations. PLDs include three distinct types of chips: PROMs, PALs, and PLAs. The most flexible device is the PLA (programmable logic array) in which both the AND and OR gate arrays are programmable. In the PROM device, only the OR gate array is programmable. In the PAL device, only the AND gate array is programmable. PLDs are programmed by blowing the fuses along the paths that must be disconnected.

FPGAs and CPLDs are classes of PLDs.

post-synthesis simulation

This type of simulation is usually done after the HDL code has been expanded into gates. Post-synthesis simulation is similar to behavioral simualtion since design behavior is being checked. The difference is that in post-synthesis simulation the synthesis tool's results are being checked. If post-synthesis and behavioral simulation match, then the HDL synthesis tool has interpreted the HDL code correctly.

primitive

A basic logic element, such as a gate (AND, OR, XOR, NAND, or NOR), inverter, flip-flop, or latch.

A logic element that directly corresponds, or maps, to one of these basic elements.

programming

Programming is the process of configuring the programmable interconnect in the FPGA.

PROM

A PROM is a programmable read-only memory.

PROM file

A PROM file consists of one or more BIT files (bitstreams) formed into one or more datastreams. The file is formatted in one of three industry-standard formats: Intel MCS86 HEX, Tektronics TEKHEX, or Motorola EXORmacs. The PROM file includes headers that specify the length of the bitstreams as well as all the framing and control information necessary to configure the FPGAs. It can be used to program one or more devices.

propagation

Propagation is the repetition of the bus attributes along a data path so that they only need to be defined on one bus in a data path.

pull-down resistor

A pull-down resistor is a device or circuit used to reduce the output impedance of a device, often a resistor network that holds a device or circuit output at or less than the zero input level of a subsequent digital device in a system.

pull-up resistor

A pull-up resistor is a device or method used to keep the output voltage of a device at a high level, often a resistor network connected to a positive supply voltage.

R

RAM

Random Access Memory (RAM) is a read/write memory that has an access time independent of the physical location of the data.

RAM can be used to change the address values (16^1) of the function generator it is a part of.

readback

Readback is the process of reading the logic downloaded to an FPGA device back to the source. There are two types of readback.

A readback of logic usually accompanied by a comparison check to verify that the design was downloaded in its entirety.

A readback of the states stored in the device memory elements to ensure that the device is behaving as expected.

register

A register is a set of flip-flops used to store data. It is an accumulator used for all arithmetic operations.

resistance

The property — based on material, dimensions, and temperature of conductors — that determines the amount of current produced at a given difference in potential. A material's current impedance that dissipates power in the form of heat.

The drive of the output pins on a network.

resistor

A resistor is a device that provides resistance.

ROM

Read Only Memory (ROM) is a static memory structure that retains a state indefinitely, even when the power is turned off. It can be part of a function generator.

routing

Routing is the process of assigning logical nets to physical wire segments in the FPGA that interconnect logic cells.

RPM

A Relationally Placed Macro (RPM) defines the spatial relationship of the primitives that constitute its logic. An indivisible block of logic elements that are placed as a unit into a design.

RTL

Register Transfer Level

S

schematic

A schematic is a hierarchical drawing representing a design in terms of user and library components.

script

A script is a series of commands that automatically execute a complex operation such as the steps in a design flow.

SDF (standard delay format)

Standard Delay Format (SDF) is an industry-standard file format for specifying timing information. It is usually used for simulation.

seed

A seed is a random number that determines the order of the cells in the design to be placed.

set/reset

This operation is made possible by the asynchronous set/reset property. This function is also implemented by the Global Reset STARTUP primitive.

shift register

A shift register is a register in which data is loaded in parallel and shifted out of the register again. It refers to a chain of flip-flops connected in cascade.

signal

A signal is a wire or a net. See “net.”

simulation

Simulation is the process of verifying the logic and timing of a design.

skew

Skew is clock delay. See clock skew.

slew rate

The slew rate is the speed with which the output voltage level transitions from +5 V to 0 V or vice-versa. The slew rate determines how fast the transistors on the outputs change states.

slice

Two slices form a CLB within Virtex and Spartan-II families.

speed

Speed is a function of net types, CLB density, switching matrices, and architecture.

STARTUP symbol

The STARTUP symbol is a symbol used to set/reset all CLB and IOB flip-flops.

state

A state is the set of values stored in the memory elements of a device (flip-flops, RAMs, CLB outputs, and IOBs) that represent the state of that device at a particular point of the readback cycle. To each state there corresponds a specific set of logical values.

state machine

A state machine is a set of combinatorial and sequential logic elements arranged to operate in a predefined sequence in response to specified inputs. The hardware implementation of a state machine design is a set of storage registers (flip-flops) and combinatorial logic, or gates. The storage registers store the current state, and the logic network performs the operations to determine the next state.

static timing analysis

A static timing analysis is a point-to-point delay analysis of a design network.

T

TAEngine

A program that performs static timing analysis on a successfully implemented Xilinx CPLD design (VM6).

TCL

Tool Command Language (Tcl) is an easy to use scripting language and an industry standard popular in the electronic design automation (EDA) industry.

TEKHEX (Tektronix)

TEKHEX (Tektronix) is a PROM format supported by Xilinx. Its maximum address is 65 536. This format supports PROM files of up to (8 x 65 536) = 524 288 bits.

testbench

An HDL netlist containing test vectors to drive a simulation.

threshold

The threshold is the crossover point when something occurs or is observed or indicated. The CMOS threshold and TTL threshold are examples.

timing

Timing is the process that calculates the delays associated with each of the routed nets in the design.

timing simulation

This type of simulation takes place after the HDL design has been synthesized and placed and routed. The purpose of this simulation is to check the dynamic timing behavior of the HDL design in the target technology.

Use the block and routing delay information from the routed design to assess the circuit behavior under worst-case conditions.

timing specifications

Timing specifications define the maximum allowable delay on any given set of paths in a design. Timing specifications are entered on the schematic.

TRACE

Provides static timing analysis of a design based on input timing constraints.

trace information

Trace information is a list of nodes and vectors to be simulated in functional and timing simulation. This information is defined at the schematic level.

transistor

A transistor is a three-terminal semiconductor device that switches or amplifies electrical current. It acts like a switch: On is equal to 1, and Off is equal to 0.

trimming

Trimming is the process of removing unconnected or unused logic.

tristate (3-state)

A 3-state, or 3-state buffer, is a buffer that places an output signal in a high-impedance state to prevent it from contending with another output signal.

tristate (3-state) condition

A 3-state condition is a high-impedance state. A 3-state can act also as a normal output; i.e. it can be on, off, or not connected.

truth table

A truth table defines the behavior for a block of digital logic. Each line of a truth table lists the input signal values and the resulting output value.

TSIM

A program that formats implemented CPLD design (VM6) files into a format usable by the NetGen timing simulation flow.

TTL

TTL, or transistor-transistor logic, is a technology with specific interchange (communication of digital signals) voltages and currents. Other technologies include ECL, MOS, and CMOS. These types of logic are used as criteria to classify digital integrated circuits.

U

unbonded

Unbonded describes an IOB used for internal logic only. This element does not have an external package pin.

Unified Libraries

A set of logic macros and functions that are used to define the logic of a design. The elements are compatible across families and schematic and HDL editors.

V

VCC pin

The VCC pin is Power (5 volts). It is the supply voltage.

verification

Verification is the process of reading back the configuration data of a device and comparing it to the original design to ensure that all of the design was correctly received by the device.

Verilog

Verilog is a commonly used Hardware Description Language (HDL) that can be used to model a digital system at many levels of abstraction ranging from the algorithmic level to the gate level. It is IEEE standard 1364-1995. Verilog was originally developed by Cadence Design Systems and is now maintained by OVI.

A Verilog file has a .v extension.

VHDL

VHDL is an acronym for VHSIC Hardware Description Language (VHSIC an acronym for Very High-Speed Integrated Circuits). It can be used to describe the concurrent and sequential behavior of a digital system at many levels of abstraction ranging from the algorithmic level to the gate level. VHDL is IEEE standard 1076-1993.

A VHDL file has a .vhd or .vhdl extension.

VITAL

VITAL is an acronym for VHDL Intitiative Toward ASIC Libraries. It is a VHDL-library standard (IEEE 1076.4) that defines standard constructs for simulation modeling, accelerating, and improving the performance of VHDL simulators.

W

wire

A wire is a net or a signal. See net.

X

xtclsh

Xilinx Tcl Shell (xtclsh), which presents the xtclsh prompt used for running scripts and using the Xilinx Tcl commands.